UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE FÍSICA

Development of Sparse Coding and Reconstruction Subsystems for Astronomical Imaging

João Maria Felner Rino Alves Silvestre

Mestrado Integrado em Engenharia Física

Dissertação orientada por:
André Moitinho de Almeida
Alberto Krone-Martins

2019

Abstract

Compressed sensing (CS) is a revolutionary signal processing technique that allows us, under a specific set of conditions, to fully reconstruct an under-sampled signal[1, 2, 3, 4]. Very early, from its inception, in 2006, to subsequent development and propagation in the following years[5, 6, 7] compressed sensing enabled advancements in photography[8, 9], holography[10, 11] and medical instrumentation[12, 13] among others[14, 15, 16, 17, 18]. Its application in astronomy however, even though some calls to action have recently been made[19], has failed to leave the test bed.

Continuing from the work developed by Bandarra and Pires in their respective Masters' dissertations[20][21], advancements will here be described on the development of a physical, out of the table, instrument; a compressed sensing astronomy camera (COSAC). Such an instrument was projected to be constituted by five subsystems: optomechanics (see [21]), signal coding, acquisition electronics (see [20]), signal reconstruction and the mechanical structure (casing and inner supports for the aforementioned subsystems). The present work focused on the development/implementation of signal coding and reconstruction subsystems, while a simple prototype for the mechanical structure is also proposed to enable testing the instrument in a real world setting; this required the redesign of the optomechanical supports. Additionally, some changes were made to the acquisition electronics in order to not only improve its behavior, but to also facilitate its integration with the signal coding subsystem; as a result two working circuit are proposed, one using an ADC of 10 bits resolution, the other an ADC of 24 bits .

A central component to this instrument, which bridges the optomechanics, signal coding and acquisition subsystems, is a digital micromirror device (DMD)[22], an array of independently controlled micromirrors which can be tilted in two, opposed, directions. Such a device can, and is, thus used to manipulate light. For this project a DLP LightCrafter, a projector development kit by Texas Instruments[23] which includes a DMD, was used to encode light signals.

The signal coding subsystem is constituted by the LightCrafter and two programs: one written in C/C++ to run either on a PC (`DMD-CS.cpp`), which main purpose is to control the DMD and communicate with the LightCrafter's processor, and which also communicates with an Arduino micro-controller that manages the acquisition electronics; the second (which is also part of the acquisition subsystem) in Arduino programming language, to run on the micro-controller (`pIDDO.ino`), which will manage the processes required to perform measurements with the electronics and communicate with the C/C++ program; the interactions between both programs are crucial to ensure synchronism between the signal coding and acquisition subsystems.

The chosen encoding basis are squared Hadamard matrices[24] that can be attained by following simple algorithms[25]; rows of such matrices were then manipulated into tilt configurations for the micromirror grid; the set of rows used will constitute a sampling matrix. Each program outputs a file, one holding information about the sampling matrix used, the other holding the measurements.

The signal reconstruction subsystem is another program that takes the files generated by `DMD-CS.cpp` and `pIDDO.ino` to reconstruct the original signal by implementing a Matlab script written by Romberg[26]. The program then outputs a BMP image file of that reconstruction.

The components of the prototype structural subsystem and optomechanical supports were designed using computer assisted design (CAD) software, with which finite element simulations were also performed to ensure those same components would be able to endure real world conditions. Some of these components were bought most of them were fabricated in the laboratory.

All subsystems were individually tested, as well as in couples (when relevant). After passing those tests, these subsystems were assembled to form COSAC. The instrument was calibrated, analysed and validated, using both versions of the acquisition circuit, in a laboratory setting with controlled lighting conditions. Comparative results of COSAC's performance for three modes of acquisition (raster, Hadamard transform optics and CS with Hadamard base) are also presented.

COSAC was shown to be able to produce images of CS measurements, performed in the visible spectrum, with at least 64×64 pixels.

Keywords: astronomy, compressive sensing, COSAC, DMD, Hadamard.

Resumo

Amostragem comprimida (CS) é uma técnica revolucionária de processamento de sinal que nos permite recuperar completamente toda a informação de um sinal contornando os limites de amostragem impostos às técnicas convencionais[1, 2, 3, 4], sendo para isto necessário que exista uma base de representação onde este sinal seja esparso, i.e. onde esse possa ser comprimido. Desde muito cedo após a sua elaboração, em 2006, até ao seu desenvolvimento e propagação nos anos seguintes[5, 6, 7] que CS possibilitou o desenvolvimento de novas abordagens em fotografia[8, 9], holografia[10, 11] e instrumentação para a medicina[12, 13], entre outros[14, 15, 16, 17, 18]. No entanto, apenas rencentemente tem sido demonstrado algum interesse em levar a aplicação desta técnica para fora do laboratório[19].

Continuando a partir dos trabalhos desenvolvidos previamente por Bandarra e Pires nas suas respectivas dissertações de mestrado[20][21], serão aqui descritos os avanços realizados neste projecto com vista ao desenvolvimento de um instrumento portátil, com aplicação prática fora do laboratório que utilize a abordagem CS; uma camara de amostragem comprimida para astronomia (COSAC). Este instrumento foi projectado para ser constituído por cinco subsistemas: óptico mecânico (ver [21]), de codificação de sinal, electrónica de aquisição (ver [20]), de recuperação de sinal e estrutura mecânica (caixa e suporte aos restantes subsistemas). O trabalho realizado focou-se no desenvolvimento/implementação dos subsistemas de codificação e recuperação de sinal, sendo ainda proposto um protótipo rudimentar para a estrutura mecânica de modo a poder integrar todos os subsistemas e assim testar o instrumento fora do laboratório; isto levou a um redesenho dos suportes óptico mecânicos. Adicionalmente, foram realizadas algumas alterações à electrónica de aquisição com a finalidade de melhorar o comportamento desta e também facilitar a integração deste subsistema com o de codificação de sinal; como resultado, são propostos dois circuitos funcionais, um utilizando o um ADC de 10 bits, o outro utilizando um ADC de 24 bits. Como subproduto do desenvolvimento destes circuitos é ainda apresentada uma *shield* com conversor de 24 bits para Arduino Uno.

A componente central deste instrumento, que estabelece ligação entre os subsistemas óptico mecânico, de codificação de sinal e o de aquisição, é um *digital micromirror device* (DMD)[22], uma rede de micro-espelhos independentes que podem assumir uma de duas inclinações opostas. Este dispositivo pode ser, e é, utilizado para estruturar luz, estando presente, actualmente, em grande parte do equipamento de projecção. Para este projecto foi utilizado um DLP LightCrafter, um projector/*kit* de desenvolvimento produzido pela Texas Instruments[23] que inclui um DMD, sendo este responsável pela codificação de sinais.

Testando a electrónica de aquisição, verificou-se que: o alcance dinâmico desta não estava totalmente aproveitado; algumas ligações realizadas entre uma das componentes e o micro-controlador Arduino, responsável pelo gestão dos procedimentos necessários à realização de medições, impossibilitavam o estabelecimento de sincronismo entre os dois; o desenho da placa de circuito impresso (PCB) não estava optimizado em relação à topologia do Arduino e às ligações necessárias entre os dois. Algumas correções aos problemas atrás mencionados são aqui propostas, tendo sido alteradas ligações e um módulo do circuito, tendo sido redesenhado o PCB de modo a este possa encaixar nos terminais do Arduino, e tendo sido acrescentado um terminal de modo a poder estabelecer comunicação `TTL` entre o Arduino e o LightCrafter.

O subsistema de codificação de sinal é constituído pelo LightCrafter, e dois programas: um, escrito em C/C++ e a ser executado num PC (`DMD-CS.cpp`), cujos propósitos são controlar o DMD, comunicar com o processador do LightCrafter e ainda comunicar com o controlador Arduino anteriormente mencionado; o segundo (que é partilhado com o subsistema de aquisição), escrito em linguagem de Arduino e a ser executado num (`pIDDO.ino`), que irá activar e desactivar as portas lógicas - dos circuito integrados (IC) presentes - necessárias a que o processo de efectuar uma medição seja realizado com sucesso, e comunicará com o PC; a interacção entre ambos os programas é essencial para garantir o sincronismo entre os subsistemas de codificação e de aquisição.

Escolheu-se, para codificar o sinal a ser medido, como bases de representação, matrizes quadradas de Hadamard[24], podendo estas ser construídas através de simples algoritmos[25]. De modo a poupar recursos computacionais, desenvolveu-se um algoritmo que recebendo como entrada o *rank* da matriz e o índice referente a uma linha desta constroi apenas essa linha. As linhas assim geradas serão posteriormente manipuladas de modo a serem utilizadas como padrões de configuração para as inclinações dos micro-espelhos, sendo que o conjunto destas linhas/padrões definirá a nossa matriz de amostragem; o método utilizado para configurar o DMD implica a transferência destes padrões codificados em formato de imagem *bitmap*[27] (BMP), pelo que foram aqui criadas as funções necessárias a manipular a informação destes padrões de modo a ser interpretada do modo desejado pelo processador do LightCrafter. Cada um dos programas previamente mencionados irá gerar um ficheiro de saída: `DMD-CS.cpp` gerará um ficheiro contendo informação sobre a matriz de amostragem; já o ficheiro gerado por `pIDDO.ino` irá conter os resultados das medições realizadas pela electrónica de aquisição. As funções e instruções escritas para gerir a comunicação entre o PC e o LightCrafter são uma implementação simplificada do código da interface gráfica deste. O programa permitirá ao utilizador definir o *rank* da matriz de Hadamard a utilizar, o número de linhas, desta, a serem geradas e o tempo que deverá demorar cada aquisição.

O subsistema de reconstrução de sinal é outro programa que, a partir dos ficheiros gerados pelos programas `DMD-CS.cpp` e `pIDDO.ino`, reconstroi o sinal real implementando um algoritmo de optimização, desenvolvido originalmente por Romberg[26]. Este programa gera um ficheiro de imagem BMP com o resultado, numa escala de tons de cinza.

As peças para o protótipo do subsistema estrutural e para os suportes óptico mecânicos foram desenhadas usando *software* de desenho assistido por computador (CAD), no qual também foram realizadas simulações de elementos finitos para garantir que tanto as peças como a estrutura são capazes de manter a sua integridade em condições reais de utilização. Algumas das peças foram compradas, as restantes foram produzidas no laboratório - tendo sido impressas em resina fotopolimérica por uma impressora 3D estereolitográfica - ou em oficinas - tendo sido maquinadas em alumínio com recurso a fresadoras de controlo numérico e de controlo numérico por computador (CNC), entre outras ferramentas. O protótipo foi desenhado tendo como objectivo ser possível anexá-lo a uma montagem telescópica equatorial comum.

Todos os subsistemas foram primeiro testados individualmente e, posteriormente testaram-se em pares: o subsistema de codificação e o de aquisição, para garantir que o encadeamento de processos entre os dois estava sincronizado; o subsistema de óptico mecânico e o de aquisição, para focar os sinais de entrada primeiro na região de interesse do DMD e, após reflexão, no detector utilizado. Os subsistemas foram depois montados na estrutura para formar o COSAC. O instrumento foi calibrado, analisado e validado, usando ambas as versões

do circuito de aquisição, em laboratório, sob condições de luminosidade controladas. São também apresentados resultados comparativos do desempenho do COSAC utilizando três modos de aquisição distintos (varrimento, óptica de transformadas de Hadamard e CS utilizando matrizes de Hadamard como base).

Foi demonstrado que o COSAC é capaz de produzir imagens, no espectro visível, resultantes de medições em modo CS com, no mínimo, 64×64 pixeis de resolução.

Palavras-Chave: amostragem comprimida, astronomia, COSAC, DMD, Hadamard.

Acknowledgments

I'd like to begin by thanking Prof. António Amorim, for proposing me the opportunity of integrating CENTRA-SIM group, in which this project was developed, and Prof. André Moitinho, for presenting me and entice me in the topic of this project. I'd also like to thank Alberto Krone-Martins for his influence, guidance, stimuli, unshakable optimism and, most of all, for always sharing his passion for science.

I'd like to thank Bruno Couto for sharing his workspace and experience, and helping me out around the workshop; Prof. Guiomar Evans, Prof. Jorge Buesco, Prof. José Augusto and Prof. Luis Peralta for lending me their help when I so required it; Izabella Hemprich for managing this project's acquisitions; João Martins for teaching me how to work with the CNC and milling machines; my colleagues, Soraia Elísio, Susana Lopes and Cláudio Filipe, for those short hallway talks where we shared our sores throughout our projects; Márcia Barros, André Silva and Pedro Garcia for allowing me to bounce around some of my frustrations; Helder Savietto and Manuel da Silva for having built the UV oven where the 3D printed parts were cured; PTRobotic's Catarina Silva and Diana Dias, for their diligence in trying to supply whatever that/whenever it was required for this project to advance.

I'd like to thank my family and friends, for bearing through all these years of growing pains and apparent detachment, I wouldn't have achieved nearly as much without their support.

Finally, I'd like to thank Rita Barreto, for providing me a reason to grow up, for always being there for me, for her patience, understanding and love, for flavoring my personal experience and allow me to fabricate some meaning to it.

"Tudo vai dar certo!"
Alberto Krone-Martins

Contents

List of Figures

List of Tables

Abbreviations

ABV - amount of base vectors used
ADC - analog to digital Converter
AM - analytical methods
BMP - bitmap
BoB - breakout board
CAD - computer assisted design
CD - card detect
CG - conjugate gradients method
CNC - computer numerical control
CS - compressed sensing
ChS - chip select
CPU - central processing unit
COSAC - compressed sensing astronomy camera
DLP - digital light processor
DMD - digital micro-mirror device
EPD - electric potential difference
ETC - estimated total cost
EUC - estimated unit cost
FEM - finite element method
FPGA - field-programmable gate array
GP - gradient projection
GPU - graphics processing unit
GUI - graphic user interface
HDMI - high-definition multimedia interface
HF - high frequency
IC - integrated circuit
IFU - integral field unit
IT - integration time
IVC - IVC102
kIDDO - imaging device with digitized output
LF - low frequency
LTC - LTC2440
LSB - least significant byte
MAD - median absolute deviation
MSB - most significant byte
NM - numerical methods

opamp - operational amplifier
PCB - printed circuit board
pIDDO - prototype imaging device with digitized output
PWM - pulse width modulation
Qtt. - quantity
RP - relative position
SC - Sylvester's construction
SPI - serial peripheral interface
SYMMLQ - symmetric linear quadratic method
TI - Texas Instruments
TTL - transistor-transistor logic
TV - total variation minimization
USB - universal serial bus
VSR - variance to signal ratio

Symbols

Greek

α - user-specified parameter to be used in Newton method iterations for residual norm calculation
β - user-specified parameter to be used in Newton method iterations to update step length
γ - projection of a signal on a basis on which it is sparse
δ - angle between a vector and the X-axis
ϵ - angle between a vector and the Y-axis
ζ - angle between a vector and the Z-axis
ν - primal vector
Λ - diagonal matrix where $\Lambda_{ii} = \lambda_i$
λ - dual vector
Φ - basis on which a signal is sparse
θ - micromirror tilt amplitude
τ - parameter that relaxes the slackness condition of the Newton method from $\lambda_i^k f_i(x^k) = 0$ to $\lambda_i^k f_i(x^k) = \frac{1}{\tau^k}$

Roman

A - known representation basis
C - $m \times N$ matrix with c_i as rows
c_i - rank N vectors
dv - dark projection
F - diagonal matrix where $F_{ii} = f_i$
f_i - linear functionals of form $f_i(x) = \langle c_i, x \rangle + d_i$
fv - flatfield projection
H - Hadamard matrix
i - enumeration index
K - sparsity of a signal
k - index of an Hadamard matrix rank, s.t. $k = \log_2 n$
l - iteration index
M - number of projections measured on the basis θ
m - number of objects to be measured
N - size of measurement vector
n - rank of a matrix

p - pixel value
T - temperature
V_{CC} - positive source voltage
V_{EE} - negative source voltage
s - step length
x - true signal
y - projection of a signal on a chosen known basis

Chapter 1

Introduction

In this chapter we'll start by describing the current limitations in traditional sampling devices, framing what compressed sensing can offer to help break those barriers. A brief description of CS is then presented, followed by examples of applications where it's been applied in the recent years and by the motivation behind behind the present work in applying CS to astronomy. We then summarize the inner workings of a CS imaging instrument and introduce its basic integrant subsystems. At last, we present a synopse of the contents of the chapters that follow.

1.1 Hypothesis

Presently, in every day life, from social affairs to scientific endeavors, most sampling devices used to record our memories, analyze our bodies or investigate our environment, are built in such a way as to abide by the Shannon-Nyquist sampling theorem [28, 29]. As a consequence of this approach, should we want to increase the resolution of our measurement we'll need to increase the sampling by the same factor. An example of this is the camera manufacturer's competition to produce cameras with an ever growing number of pixels. Issues arise, however, when we're required to share/store the files generated by these devices, as more samples will equate with more data and more data with more memory required to hold it. This has been solved by the usage of compression algorithms that are used after the data is acquired.

Compression can be achieve if the uncompressed data displays some degree of sparseness, that is, if there's some degree of correlation between the data elements. This can then be translated as a series of spikes and voids in what can be described as a reciprocal space of the data. By storing only the coefficients that hold these spikes we're effectively holding the same information (in the case of strict sparseness) while using a lesser amount of data. But if the same information can be gathered from a smaller dataset and if we're increasing the amount of data generated only to compress it later on, why not just go in another direction and try to acquire the compressed set instead? In comes compressed sensing.

1.2 Compressed Sensing

In the late 1970's a foreign measuring method started to be studied for its possible applications in optics. The core of this method lies in measuring several objects in groups instead of measuring each one individually. These measruments can be encoded into a matrix, where each row defines a measuring mask for the set of objects that, when applied to optics, will be realized by transmiting,

absorbing or reflective elements. The best matrices for this are Hadamard matrices and their derivatives, and these have traditionally been chosen for these applications (in section 3.1 a brief description of Hadamard matrices is supplied), as such this new approach to optics has been named Hadamard transform optics.

This methodology, which had previously been studied in statistics under the name of weighing designs, was shown to reduce considerably the magnitude of the errors associated to measurements by measuring combinations of the objects. Its only restriction is that if we want to measure m objects then m measurements must be made[30]. But what if this hadn't to be so?

1.2.1 Theory

In the turn of the millennium, making use of the aforementioned concept of section 1.1, a new set of ideas was developed[1, 2, 5, 6, 14] into a new signal sampling theory, compressed sampling (CS).

Unlike the classical approach, whose limits are described by the theory developed by Shannon & Nyquist, CS only requires that the signal which we want to measure is compressible in some basis, in other words, that a basis exists in which the representation of that signal is sparse.

Let x be a signal represented by a measurement vector, γ, of rank N, where M measurements are non-null and where $M \ll N$. If x is projected onto a basis, Φ, the resulting measurement vector can be calculated using eq. 1.1, from which follows that the signal is naturally compressed while being measured.

$$\gamma = \Phi^T x \tag{1.1}$$

The signal x is described as being M-sparse in Φ if γ only has $M \ll N$ non-null entries. These non-null coefficients will hold the relevant information about the signal and an approximation of x can be made using a single coefficient. While measuring only these coefficients are to be saved, as such the measurement will tend to be much smaller, in regards to the data generated, than the original signal (computationally a coefficient is considered null if it's smaller than a predefined floor value; this regime is described as non-restricted sparsity and is more likely to occur in real world applications due to the intrinsic noise of measuring procedures).

Obviously these K coefficients of γ can't be measured directly as we don't know the basis which generates them. As such, $K < N$ projections of the signal are measured instead in another set of basis (of our choosing), A, where $K \geq M$. So, for practical applications, what is actually measured are the coefficients of vector y, which can be expressed by eq. 1.2.

$$y = Ax \tag{1.2}$$

After measuring it, the signal will have to be reconstructed. Such process requires inverting a matrix, which for large rank matrices is a computationally exhaustive problem. It's therefore necessary to resort to optimization algorithms.

One such way to solve our system, should it present as an exact problem, would be using a minimization solution of an l_1-norm optimization:

$$min_\gamma ||\gamma||_1 \quad s.t. \quad y = A_\Lambda \Phi \, \gamma \tag{1.3}$$

where Λ indexes a deterministic ensemble of randomly select vector from the basis A. Under

certain assumptions on the measurement matrix, A, the solution found can be proved to belong to the solution set of l_0-minimization.

A more realistic model can be adopted by adding noise and optimization constraints to the equation.

1.2.2 Existing Applications

Applications of this theory actually predate it. In the late 1970's seismologists found that their technique to see underground (bouncing seismic waves off discontinuities in strata) was yielding better results than was thought possible at the time[7], this turned oil-prospecting into a less fortune-based affair. Soon after CS was developed, some architectures and setups started being proposed for single-pixel cameras[9, 8, 15] on the optical domain and beyond[16], while at the same time others made similar propositions in medical instrumentation[12, 13]; on the turn of the decade the first application in holography were suggested[10, 11].

In this decade research has broaden into other domains of application such as mobile phone video recording[31], communication networks[17] and electron microscopy[18].

Interest has also recently sparked among the astronomy community[19].

1.2.3 Opportunities in Astronomy

Presently, astronomy faces a few challenges regarding image acquisition. Due to development of devices with increasingly better resolution and field of view, as mentioned in section 1.1, the amount of data produced by such devices is at the petabyte scale (and may soon climb up to the exabyte scale[14]), while the cost and complexity of building such large arrays of detectors also increases. The CS approach, through the application of instruments based on the single-pixel camera, can solve both issues. Additionally, the ability to do imaging using a single detecting pixel allows for the application of this method to all wavelengths, something which is currently not possible using CCD or CMOS detectors.

1.3 A Compressed Sensing Astronomy Camera

This dissertation project is integrated in a set of projects which aim at the development of an imaging system for astronomy that follows the CS approach, a compressed sensing astronomy camera (COSAC). Such instrument shall be constituted by five subsystems: an optomechanics subsystem, a signal coding subsystem, an acquisition subsystem, a signal reconstruction subsystem, see fig. 1.1, and a structural subsystem.

The optical subsystem, developed by Pires in her Masters' dissertation[21] and which optimization is described in section 5.1.1, is a structure composed by lenses and supports for the acquisition and signal coding subsystems. Its purpose is to focus the light signal, that enters the instrument, onto the reflecting surface of the coding subsystem, and to focus the reflected signal into the sensor of the acquisition subsystem.

The acquisition subsystem is comprised by a printed circuit board (PCB), developed by Bandarra in her Masters' dissertation[20] and which optimization is described in section 2.1, which uses a photodiode for a sensor and is controlled by an Arduino microprocessor, which is in turn also connected to the light coding subsystem as to synchronously trigger both subsystems in order to successfully take samples.

The signal coding subsystem, includes a digital micromirror device[22] (DMD) and two control programs written to not only control the DMD but establish the communication between the user and the DMD and the acquisition subsystems, as to take a defined number of samples for each signal encoding. One of these programs is to be run on a computer while the other shall run on an Arduino.

The signal reconstruction subsystem is another program that'll use the complete sampling data gathered by the acquisition subsystem in one acquisition run, and run it through an optimization algorithm, as to obtain the information of the original signal. A description of the algorithm is presented in chapter 4.

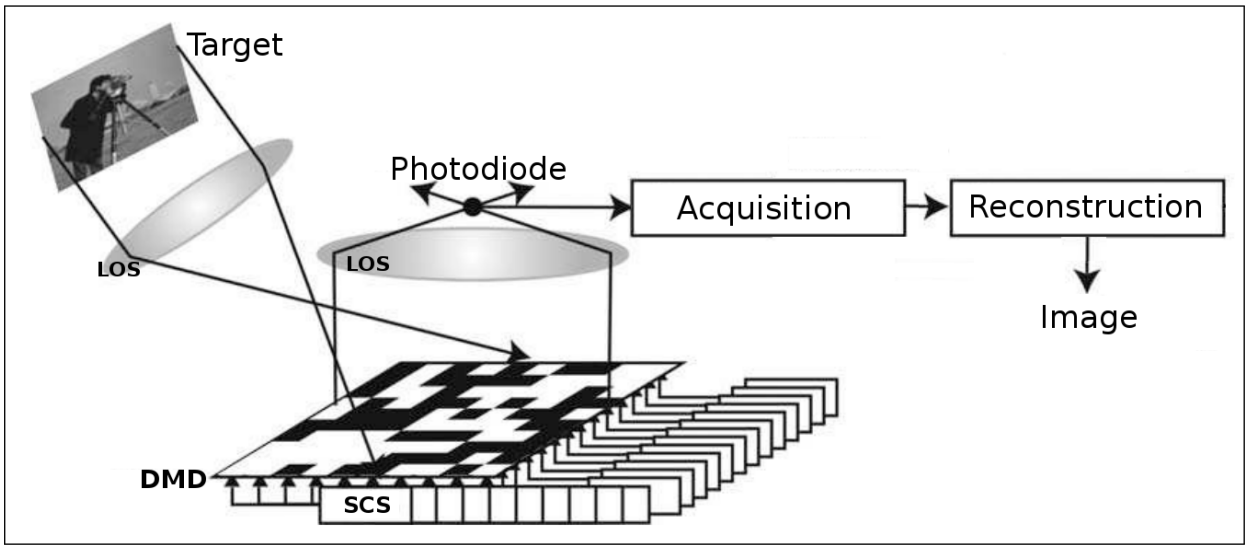


Figure 1.1: *Schematic of a CS based imaging system. LOS - lenses of the optical subsystem; SCS - signal coding subsystem. The target signal is modulated by the DMD, which is controlled by the SCS, and is reflected to the photodiode where it is collected for processing by the acquisition subsystem; the output data of the acquisition subsystem is sent to the reconstruction subsystem which, after an acquisition run, uses the samples to recover the target's information (adapted from [9]).*

At last, the structural subsystem's purpose is to allow for the testing of the system in a real-world setting, more specifically by being able to hold the other subsystems while at the same time allowing to attach the instrument to equatorial setup telescopes.

1.4 Thesis

In the next chapters the various elements developed to accomplish a functional COSAC are characterized.

Chapter 2 starts with an outline of the previous acquisition subsystem and a description of the modules that constitute the electronic circuit and their functions. Then the optimization of the acquisition, achieved through the redesign of both circuit and PCB, is detailed.

Chapter 3 is initiated with a description of how the subsystem is to work and how its constitutive parts (DLP LightCrafter and control programs) are to interact, and an introduction of Hadamard matrices as the coding basis to be used. Then the relevant aspects, to this project, of the DLP LightCrafter are presented. Next, the programs written to manipulate the DMD and interact with the acquisition subsystem are explained and the functions that were developed for those purposes are described. Finally, a test, made to ensure

that both the DMD and the control programs are working as intended, is described and the yielded results are presented.

In chapter 4 the signal reconstruction subsystem is detailed in its implementation of an l_1-minimization algorithm as a `C/C++` program. First a description of the algorithm, as well as of the code used, is made. Then simulations, as a first test to it, are conducted, and the results presented.

In chapter 5 a design for a prototype mechanical structure, for support and casing of COSAC, can be found, as well as a description of changes made to the optical subsystem to adapt it to the mechanical structure. Then a brief introduction to the finite element method is made, followed by a characterization, and results, of simulations, realized to ensure that the design of the structure behaves as intended in a given range of circumstances, are presented. The fabrication processes, for the parts that weren't bought, are then summarily described.

In chapter 6 adjustments of, and tests to, the individual subsystem to improve their integration into a functional instrument are reported. Calibration and testing of COSAC is then detailed and results are presented.

COSAC is proposed as a proof of concept for an imaging device for astronomy, that implements CS, which is ready for coupling with regular equatorial mounts, rather then being an on-table setup.

Chapter 2

Optimization of the Acquisition Subsystem

In the previous chapter we presented a general view of CS and its current and potential applications to instrumentation and signal processing, and described as well the elementary subsystems of a proposed architecture of a CS based imaging instrument. In this chapter we'll start by outlining the acquisition subsystem developed by Bandarra in her Masters' dissertation[20], and then describe the procedures, tests and implementations that were performed in order to optimize it.

The acquisition subsystem was, as mentioned earlier, developed by Bandarra and detailed in her dissertation[20], and is comprised of the kIDDO PCB (kIDDO stands for Imaging Device with Digitized Output) and a control program to run on an Arduino microprocessor, which will henceforth referred as `kiddo.ino`. This subsystem is responsible for measuring the encoded light signal, that is the projection of the signal on our chosen basis, and returning a properly modulated value as a bit stream.

kIDDO's sensor, its single pixel, is a Hamamatsu S1223[32], a photodiode that'll collect the encoded light signal and output a current, I_0. An integrating opamp, in this case an IVC102[33], henceforth referred to as IVC, takes I_0 as input and outputs an electric potential difference (EPD), V_0. This EPD we'll have to be converted to a digital signal in order to be processed after acquisition, as such an ADC was used, an LTC2440[34], henceforth referred to as LTC. The LTC shall receive as input V_1 but, because the range of V_0 is different than the range of V_1, an adaptation module, to match the range of V_0 into V_1's, is necessary; this module is comprised of a 2.5 V Zener diode and two resistors of 1.2 kΩ and 2.2 kΩ. To control the IVC and LTC processes these are connected to an Arduino Uno Rev. 3[35], the first using regular `PWM` and the second through `SPI` protocol. The Arduino then takes the bitstream output of the LTC and converts it into ready to read data which will be saved in a micro SD card, to which the Arduino is also connected via `SPI` and which is connected to the circuit by an Adafruit breakout board[36] (BoB). A 5 V regulator, an LM1086[37], is used as a supply for the LTC while the remaining integrated circuits (IC) are supplied by a symmetrical 1A power source set to supply ±15 V. Additionally, all input/output pins are coupled with capacitors to filter out low frequency (LF) and high frequency (HF) noise. For more detail kIDDO's schematics should be consulted in [20].

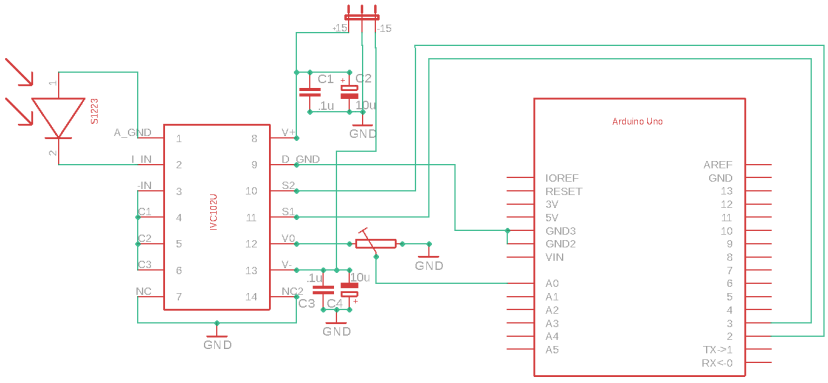


Figure 2.1: *Schematic for the IVC102's test circuit.*

2.1 Changes to the Acquisition Electronics

Upon testing an available kIDDO prototype it was realized that its functionality was limited. By consulting IVC102's and LTC2440's manuals[33, 34], the Arduino information about `SPI` protocol[38], by verifying the connections of the PCB and by making individual test circuits for the IVC and LTC in which their responses were monitored, it came to light that some of the connections that managed the `SPI` communication were incorrect and that the adaptation module was more strict than what was required (with V_1 being measured in the $[0, 1.25]\,\mathrm{V}$ interval when it could be set to $[0, 2.5]\,\mathrm{V}$. Furthermore, the design of the PCB didn't correctly account for the geometry of the Arduino, resulting in an unbalanced mount when assembled.

Prompted by these issues, changes the circuit and PCB were made and are addressed in sections 2.1.2 and 2.1.4 respectively. Some changes were also made to the acquisition control code running on the Arduino, these shall be addressed in sections 6.1.1 and 6.1.1.

2.1.1 Testing of kIDDO's Integrated Circuits

Due to the different problems and limitations identified witht he kIDDO board, it was decided to individually test both the IVC102 and the LTC2440; to ease this task, these IC's were soldered to breakout boards. Test circuits were then assembled on breadboards and connections were established with an oscilloscope (Tektronix TDS 2014) and an Arduino Uno, to compare the IC's behavior with what is referenced in their documentation[33, 34] and make readings, respectively.

Testing IVC102

To test the IVC: an Hamamatsu S1223 was connected to its input channel; IVC's -In, C_1, C_2 and C_3 were shorted; since the Arduino's ADC's conversion range is between $[0, 5]\,\mathrm{V}$ and the IVC's output range is referenced at $[0, 14.7]\,\mathrm{V}$, a connection was made from the IVC's V_0 to Arduino's A0 pin using a $5\,\mathrm{k\Omega}$ potentiometer, which was regulated so as to match both intervals' limits; the integration proccess is controlled via two switches (S_1 and S_2), the IVC's pins corresponding to these switches were connected to I/O pins of the arduino (pin 3 and 2, respectively); at last, a program was written to both control S_1 and S_2, and readout V_0 (the code can be found in appendix C.8). The previously mentioned IVC's pins were also probed using the oscilloscope. A schematic for the above described connections can be seen in fig. 2.1.

Using a led lantern, different luminosity levels, step-like, were shone into the photodiode's

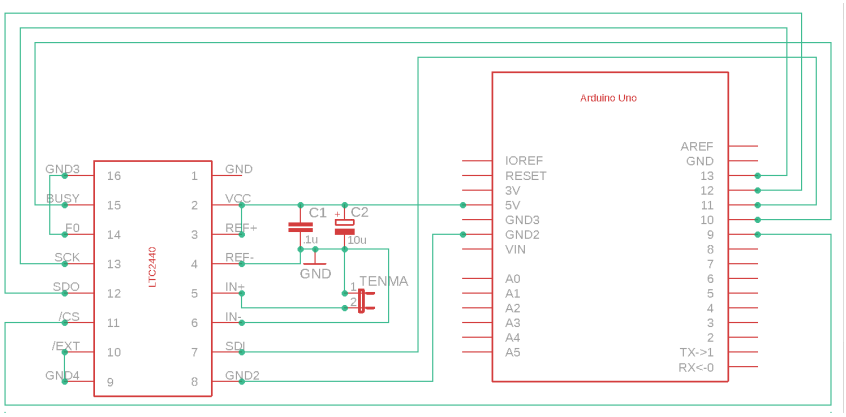


Figure 2.2: *Schematic for the LTC2440's test circuit.*

window, so as to check the IVC's output response. The behavior of the three pins was monitored and compared with what was described in fig. 3b of its manual[33], and they were found to be working as expected. Additionally, the Arduino's A0 pin that was used was also probed, and it displayed the same shape as V_0 though on different range.

As a subproduct of this test a lower resolution version (Arduino's ADCs have 10bits) of the acquisition electronics was created. A PCB was designed (see section 2.1.4) and produced so as to fit as a shield for an Arduino Uno, this shield was named pIDDO-10bit (prototype imaging device with digitized output).

Testing LTC2440

To test the LTC: `SPI` communication pins were connected as recommended, LTC's SCK to Arduino's pin 13, LTC's SDO to Arduino's pin 12, LTC's SDI to Arduino's pin 11; LTC's BUSY pin was connected to Arduino's pin 10; LTC's ∼CS pin was connected to Arduino's pin 9; LTC's V_{CC} and REF^+ were connected to Arduino's 5V; LTC's IN^+ was connected to the output of a power source (TENMA R 72-2535); the remaining pins of the LTC, as well as the power source's ground, were shorted to the Arduino's ground; finally, a program for the Arduino, to control the conversion and interpret the result outputted by the LTC, based on code writen by Beale[39], was written and can be found in appendix C.10. A schematic for the above described connections can be seen in fig. 2.2.

Through the Arduino's serial monitor it was possible to check the readout result and compare it to both the power source's monitor and measurements performed with a digital multimeter (KEYSIGHT 34461A). The digital multimeter was here used to probe the SCK, SDO and BUSY pins. It was, in this way, verified that the LTC's behaviour was compliant to the manufacturer's description.

As a subproduct of this test a 24bit ADC shield for Arduino Uno was created. A PCB was also designed (see appendix A.3) but, since it was of no consequence for the rest of this project, it wasn't produced.

2.1.2 Redesiging kIDDO

Starting from the test circuits previously described, the redesign of the kIDDO board came down joining those two circuits and performing a couple of adaptations.

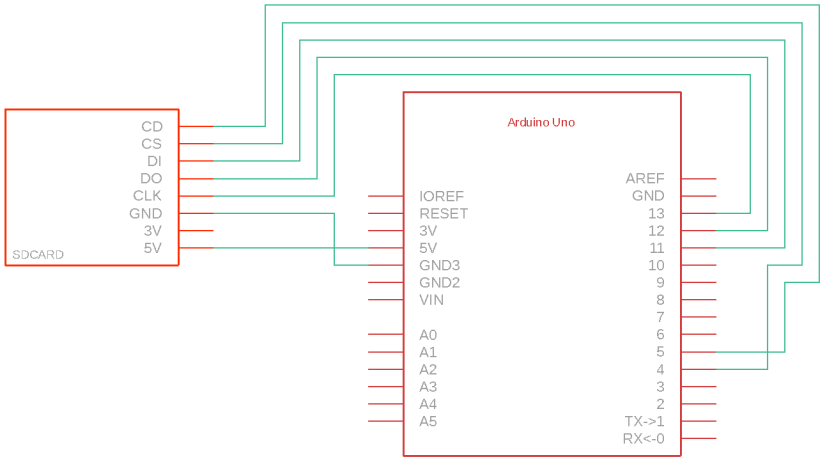


Figure 2.3: *Schematic for the connections between the SD card breakout board and the Arduino Uno.*

Since the dynamic range of the LTC can cover the interval of [0, 2.5] V, instead of the [0, 5] V provided by Arduino Uno's ADC, the potentiometer had to be adjusted in order to match the IVC's output range to this new interval. Moreover, upon establishing this connection with the LTC's IN^+ the output range of the IVC was reduced to approximately [0, 11] V, as such the potentiometer was adjusted while probing IVC's V_0: a led lantern was used to saturate the IVC's output while the LTC's response was monitored using Arduino's serial monitor; the potentiometer was adjusted until LTC's maximum output response was observed. Additionally, instead of supplying the V_{CC} and REF^+ pins with the Arduino's 5V pin, the same LM1086 of the original design was used so as to have a more stable source.

The connections described in section 2.1.1 ensure that synchronism between the LTC and the Arduino can be established within the context of `SPI` communication with the Arduino as the master and the LTC as the slave. Likewise, as a second slave, BoB was also connected to the Arduino: CLK to pin 13; DO to pin 12; and, DI to pin 11. To ensure that there's no message mixing, BoB's CS was connected to Arduino's pin 4. Finally, BoB's CD, 5V and GND pins were connected to Arduino's pin 5, 5V and GND, respectively. BoB's 3V pin was left floating. These connections were also added to pIDDO-10bit. A schematic of BoB's connections to the Arduino can be seen in fig. 2.3.

With the main modules functioning properly, capacitors of 0.1 μF and/or 10 μF, were added to the supply input pins of each IC, following the manufacturers' recommendations[33, 34, 37], to bypass LF and HF noise. A 100 μF capacitor was coupled to the output pins of the LM1086s as recommended in [37]. Additionally, a 0.1 μF capacitor was coupled to LTC2440's IN^+ pin to improve its performance.

Using, once again, a led flashlight to induce step-like luminosity variations on the detection window of the photodiode, the circuit was tested and it was verified to appear to be functioning properly. The resulting circuit was named pIDDO-24bit and its schematic can be consulted in appendix A.2.2.

2.1.3 Testing the Linearity of LTC2440

Given that the data shall be conveyed through the LTC, it is of the utmost necessity that its dynamic range can display a good level of linearity. To test its linearity the circuit described

above was used, with a test program having been written to run on the Arduino solely for this purpose (the code can be consulted in appendix C.11). This program, on each iteration of its main loop, makes a series of acquisitions. For each series the Arduino gradually increments the integration time (IT) for the IVC until the IT is high enough that the IVC becomes saturated, and for each integration time it reads the LTC's output and, since it's a 24bit binary output, converts it to a number within the defined dynamic range (in this case [0, 2.5] V). The instructions to control the LTC are based on those written by Beale[39].

Because the laboratory where this test was done couldn't supply steady and constant lighting conditions, the test setup was left to work for 5 hours; within that period 210 series of acquisitions were made. These 210, due the aforementioned lighting conditions, were then categorized into sets according to the IT needed to saturate the IVC, or the number of acquisitions made in each series. This sets were then purged of those that presented less than 10 series. For the remaining sets was calculated the average, median, standard deviation and median deviation for each IT across series within the set.

The results of this test were more satisfying than expected, considering the lighting conditions, with the LTC displaying an almost perfectly linear behavior across its dynamic range. The results can be observed in fig. 2.4.

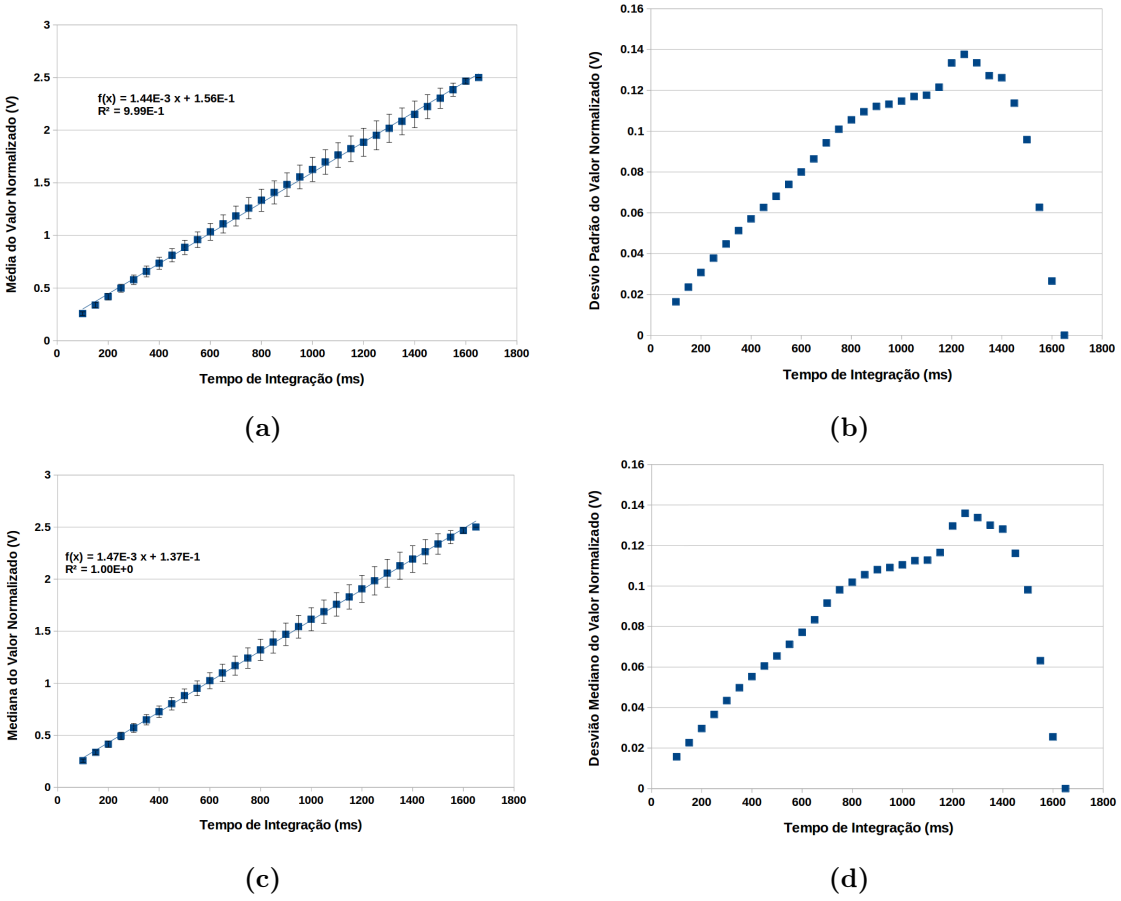


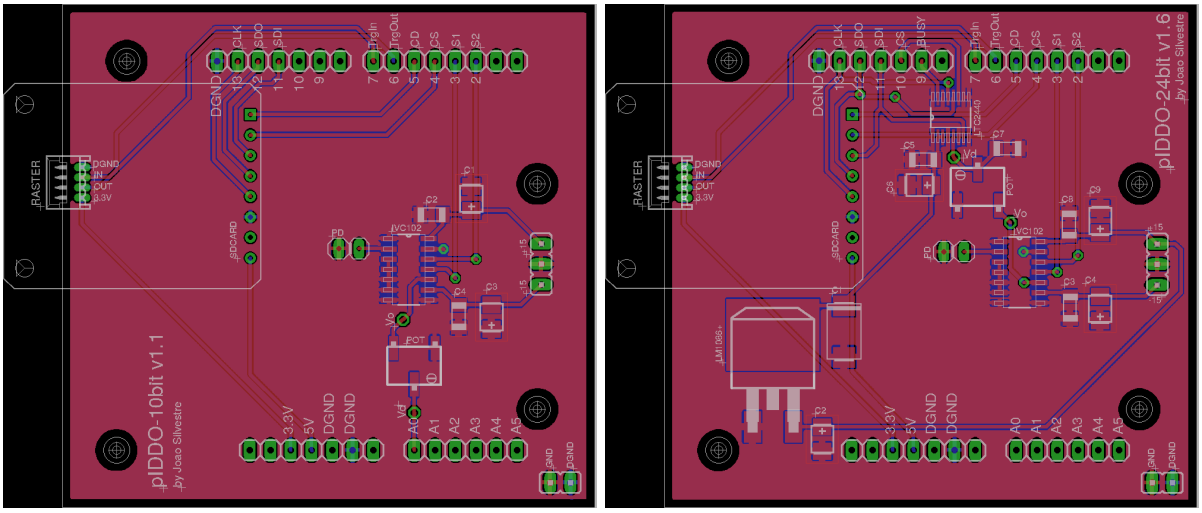
Figure 2.4: *Combined linearity test results for the most populated set of acquisition series. On the Y-axis is represented the output value of the LTC, normalized from binary to its dynamic range; on the X-axis the integration time is presented. In fig. 2.4a we've the average of the output value, in fig. 2.4b its standard deviation, in fig. 2.4c the median of the output value and in fig. 2.4d the median deviation.*

2.1.4 pIDDO-24bit PCB

The new PCB, prototype Imaging Device with Digitized Output (pIDDO), was designed with the topography of the Arduino in mind, with the purpose of connecting pIDDO as an Arduino shell, its dimensions and spacing holes matching those of the Arduino. A 4 pin raster signal socket was added to PCB design, so as to serve as a TTL communication gate between the Arduino and the signal coding subsystem, as well as a 1×2 and a 1×3 wire-to-board connectors and two ground pin-holes, one analog and the other digital, to be later connected using a wire. TTL's IN and OUT pins were connected to Arduino's pins 7 and 6, respectively. Additionally, to facilitate debugging procedures, some holes for probing pins were placed on the following vias of interest: IVC's V_0 pin, V_d via, and LTC's IN^+ pin.[1]

pIDDO was designed, with EAGLE, using two layers with one surface serving as the analog ground and the other as the digital ground. The same software was used to generate the gerber files that were handed to the company iFastPCB which handled the fabrication of the PCBs. An overlap of both layers can be checked in fig. 2.5, while separate images of both layers and a schematic can be consulted in appendix A.

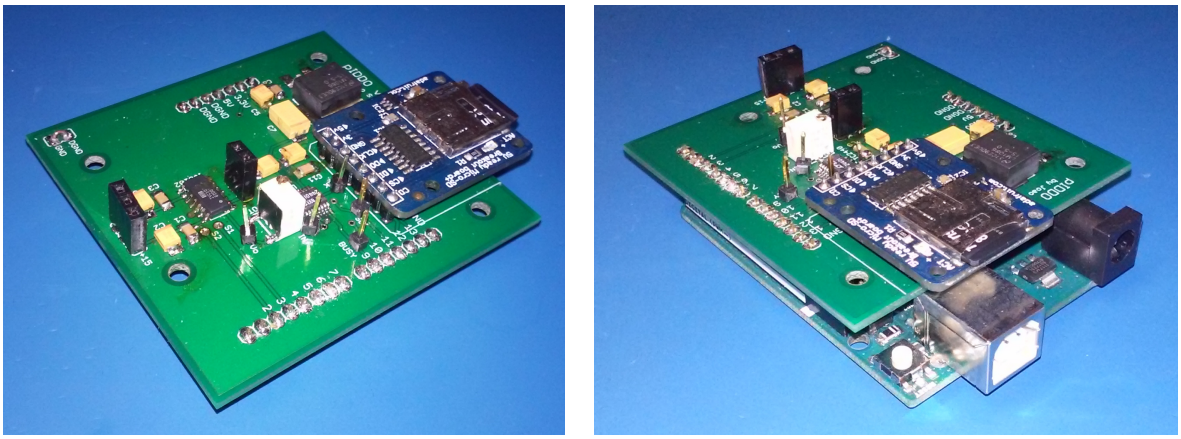
Prior to soldering the electronic components the connections were tested for both continuity and correctness according to the projected schematics and PCB design. The electronic components (bought from PTRoboticss Unipessoal Lda) were then soldered in the laboratory using a Weller R WXD 2 soldering station and, afterwards the connections were again tested for continuity and short circuits. In fig. 2.6 an example of a produced PCB and a PCB-Arduino assembly is shown using a earlier version of pIDDO-24bit.



(a) *pIDDO-10bit* **(b)** *pIDDO-24bit*

Figure 2.5: *PCB's top and bottom layers. The analog ground is presented in red and the digital ground in blue.*

[1] The same considerations were applied to pIDDO-10bit's PCB design except for the probing hole regarding the LTC.



(a) **(b)**

Figure 2.6: *In fig. 2.6a the soldered components of pIDDO-24bit v1.4 can be seen, while in fig. 2.6b the pIDDO-Arduino assembly is displayed.*

Chapter 3

Development of a Signal Coding Subsystem

Before a light signal to be measured reaches the acquisition subsystem, it is conducted by the optical subsystem to the signal coding subsystem. This last subsystem is comprised of a DLP LightCrafter development kit and two control programs, one written in `C/C++` running on a PC, with which the user can interact, through the console, and another running on an microcontroller, which is also responsible for the control of the acquisition electronics and will interact with the first program so that the coding and acquisition processes are properly synchronized.

The encoding of the light signal will performed by a digital micromirror device (DMD), which is included in the DLP LightCrafter. The DMD is a grid of micromirrors that, by reflection, can effectively modulate an incident signal. To make use of the DMD included in the DLP LightCrafter this device's optics and leds were removed, leaving the DMD exposed.

As described in 1.2.1 to encode a signal we wish to measure, we must first chose a basis upon which to project said signal. For this application we chose the Hadamard family of matrices due to the simplicity associated to their construction. Each line of an Hadamard matrix will then be transformed into a map that'll be used to configure the DMD.

3.1 Of Hadamard Matrices

Hadamard matrices, named after Jacques Hadamard[24], are square matrices whose entries are ± 1 and whose rows are mutually orthogonal. These matrices have applications in error correction codes[40], estimation of the variance of statistical estimators[41], spectrometry[42], among others, and their use has been found useful in signal processing for CS, as basis to encode the signal.

3.1.1 Sylvester's Construction

The first record of a construction algorithm for Hadamard matrices belongs to James Sylvester[25], and dates back to 1867. This algorithm, henceforth referred to as Sylvester's Construction (SC), is described below.

Let us consider H_n an Hadamard matrix of $rank\{H_n\} = n$, we can then get the matrix H_{2n} as the partitioned matrix of eq. 3.1.

$$H_{2n} = \begin{bmatrix} H_n & H_n \\ H_n & -H_n \end{bmatrix} \tag{3.1}$$

Let

$$H_1 = \begin{bmatrix} 1 \end{bmatrix}$$

It's then trivial to build the series of matrices described by eq. 3.2.

$$H_{2^k} = \begin{bmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{bmatrix}, \quad \forall k \in \mathbb{N} \tag{3.2}$$

Though conceptually simple, the execution of this algorithm becomes computationally intensive with the increase of K as such, in section 3.3.1, patterns in the results of this algorithm are explored with the intent of creating an algorithm capable of creating single rows of SC's Hadamard matrices instead of the matrices themselves. This new algorithm will later be used in the process of creating the DMD's configuration maps.

3.2 The DLP Lightcrafter

The DLP LightCrafter[43] is a device that aims at the integration of light projection in industrial, medical and scientific applications. The LightCrafter is equipped with a 0.3 WVGA chipset which allows the projection of structure light, smart illumination and wavelength selection[44].

Using its graphic user interface (GUI), a user can create, save and project pattern sequences. This application mediates the interaction between the user and a TMS320DM365 (DM365) processor[45], with an ARM9 DSP kernel, which runs a Linux OS (this allows the user to develop software around the chipset). The LightCrafter can also be configured to use output or input triggers should synchronization with sensors, control systems or cameras be required. These triggers are managed by the LightCrafter's FPGA.

Still relevant to the work that shall be described, from section 3.3 until the end of this chapter, is that the LightCrafter is equipped with a DLP3000, a digital micromirror device (DMD) that shall be described in section 3.2.1.

A block diagram of the LightCrafter can be seen in fig. 3.1.

3.2.1 The DLP3000 DMD

The DLP3000 is comprised of a 608×684 diamond grid of square micromirrors (see fig. 3.2). Each micromirror measures $10.8\,\mu$m diagonally and can tilt $\pm 12°$relatively to grid plane. The DMD is also highly efficient in visible light (420 to 700 nm) displaying window transmission of 97%, micromirror reflectivity of 88%, array diffraction efficiency of 86% and array fill factor of 92%[44].

The diamond grid allows different resolutions up to an aspect ratio of 854×480 and can be manipulated through the digital controller DLPC300[46]. This controller is responsible for mapping patterns (input data) into the micromirror grid and is capable of switching between a sequence 96 1 bit, monochromatic patterns at a maximum of 4 kHz; ramping up the memory usage to a sequence of 1024 of the same kind of patterns drops the maximum switch rate to 300 Hz[43].

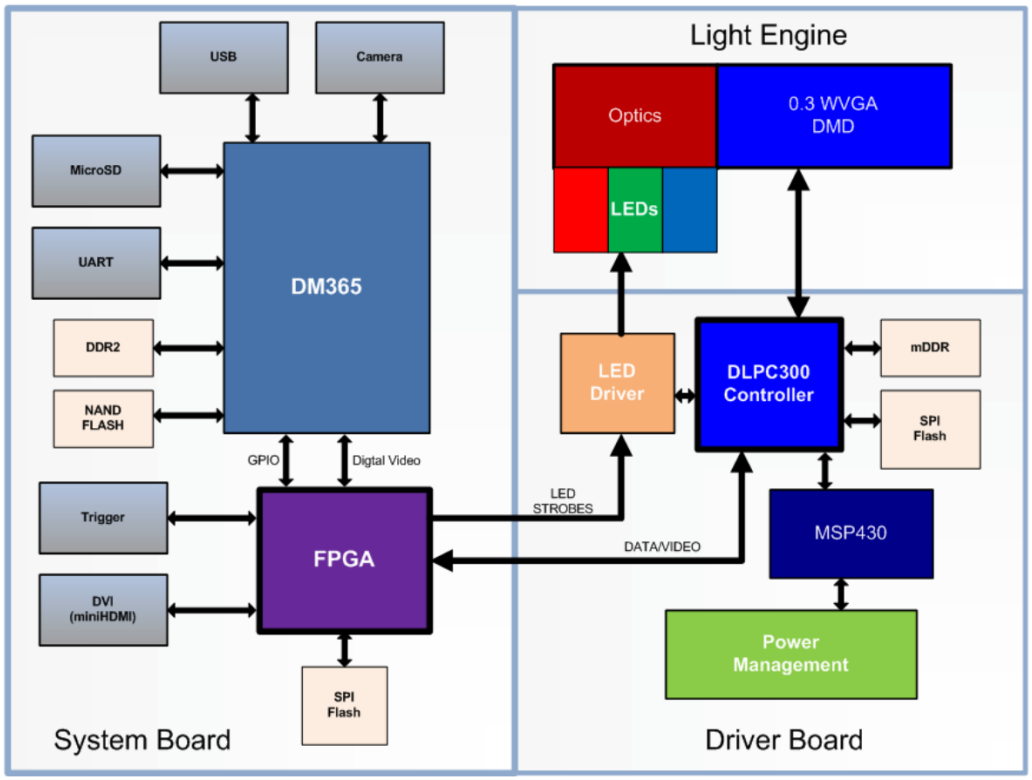


Figure 3.1: *Block diagram of the DLP LightCrafter (taken from [43]).*

Considering the grid's configuration it's clear that to go from a generated data array to a loaded pattern more steps will be required. This data will need to undergo manipulation that in image terms will correspond to an offset, so that the pattern sits in the middle of the grid, and a rotation, to avoid deformation and achieve an accurate representation on the grid, this last manipulation comes however with a cost. Consider a square of 2^k pixels to be accurately represented in a diamond grid of $A \times B$ pixels, the conditions expressed in eq. 3.3 must be met.

$$\begin{aligned} 2^{k+1} - 1 \leq A \\ 2^{k+1} - 1 \leq B \end{aligned} \tag{3.3}$$

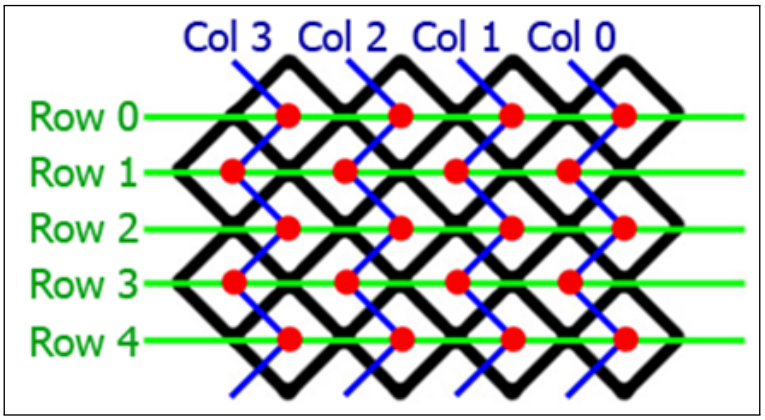


Figure 3.2: *Layout of the configuration and indexation of the DMD's micromirror grid (taken from [48]).*

Since in the present case $A = 608$ and $B = 684$ we have that $k_{max} = 8$, that is, the maximum

number of pixels the generated patterns will have is 256×256. This yields a 84% of unused micromirrors in the grid. However large that percentage may be, in this application accuracy is imperative and as such this manipulation shall be used.

A detailed explanation of these manipulations can be found in section 3.3.2.

3.2.2 The PC-DM365 Communication Protocol

As previously stated, PC-DM365 communication is established via a USB port through packet exchange. These packets are nothing but byte arrays that with a structure that shall now be described and can be seen in fig.3.3.

- Byte 0 (B_0) - describes the type of the packet. The possible descriptions are: busy, error, write, write response, read, read response.
- B_1 and B_2 - are respectively the MSB and LSB of a 16bit command type code loaded on the packet, this code is associated to an instruction that will tell how to handle the data content of the packet (should there be any). Several of these are available and a detailed list can be found in [45], however in this application only 9 were used, them being:
  - `0x0100` - reads the software and firmware versions of the LightCrafter modules;
  - `0x0101` - reads/sets the current display mode which can be one of the following: static image, internal test pattern, HDMI video input, reserved, pattern sequence display;
  - `0x0105` - loads a static bitmap file into the LightCrafter's DLPC300 memory buffer;
  - `0x0400` - defines/reads a pattern sequence setting such as the type of pattern, how many patterns are to be loaded on the sequence, the type of trigger to be used to advance the sequence, etc.
  - `0x0401` - defines/reads one pattern of the pattern sequence. In this case the data content of packet will be the number corresponding to the position of the pattern in the sequence and a BMP file's data.
  - `0x0402` - starts or stops the display of currently defined pattern sequence.
  - `0x0403` - advances the pattern to the next stored pattern. This is only valid in SW Trigger mode. No data payload is used.
  - `0x0480` - defines/reads an extended pattern sequence setting (similar to `0x0400`)
  - `0x0481` - defines/reads one pattern of an extended pattern sequence (similar to `0x0401`)
- B_3 - flag that indicates whether the present packet contains all the data associated with the command expressed in B_1 and B_2 or not and, if not, if it's the first, an intermediate or the last packet of a packet stream.
- B_4 and B_5 - are respectively the LSB and MSB of a 16bit number used to discriminate the size of the data load in the packet, being that value limited to $2^{16} - 1$ bytes per packet.
- B_6 ... B_{N-1} - data load, ;
- B_N - control byte which value is calculated by the `checksum()` function which shall be described in section 3.3.2.

The packet is in this way structured in three parts, the header (bytes 0 through 5), the load (bytes 6 through N-1) and the checksum (byte N).

HEADER						DATA	CHECKSUM
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6...Byte N	Byte N+1
Pkt type	CMD 1	CMD 2	Flags	Payload length		Data payload	Checksum

Figure 3.3: *Schematic of the structure of a data packet to be used to communicate with the DM365.*

3.3 Signal Coding Program

The signal coding subsystem is controlled by functions written to implement the communication structure explained in the previous section and others to manipulate the data that is to be sent use within the packets. Additionally, in order to make the subsystem as autonomous as possible this program must also communicate with the acquisition subsystem so that it knows when to and not to perform a new manipulation of the DMD. For this some changes to the code running on the Arduino were made, these will be detailed in section 6.1.1.

The functions that handle the communication directly (packet construction, packet trade and connection port structure) are based on those present on the LightCrafter's graphic user interface (GUI) build files; these functions are described in section 3.3.2.

The remaining functions and structures were created by reading the information available in [43, 45, 46, 47], consulting with members of the TI's E2E™ Community[48] as well as good deal of learning provided by trial and error.

In this section an algorithm developed to generate a row of an Hadamard matrix that could be constructed using the SC will be presented (in section 3.3.1), followed by a description of all functions created to be used in this control program (in section 3.3.2). Next, in section 3.3.3, the main code to run on PC is outlined and this section is ended by an overview of the results attained during testing the program's performance.

The program will manipulate arrays into a packet's structure. Packets carrying in their load data formated into a BMP image pattern will be used to change the state of each of the DMD's micromirror. Because the DLP LightCrafter is a projector, testing whether the manipulation of the micromirror grid was successful or not was easily achieved by simply displaying the content loaded in those packets. That pattern is nothing but an Hadamard matrix row that was manipulated and adjusted into being accurately represented as a square matrix in the diamond grid. Along the exposition that will follow within this section the terms "pattern" and "display" shall be used to summarily describe "an Hadamard row, whose elements have been sectioned in order to construct a square matrix, that has been stored in an array whose positions have been switched around as to accurately represent the previously mentioned matrix in a diamond grid" and "loading the BMP pattern so that the micromirror grid configuration is changed to match that pattern", respectively.

3.3.1 Generating One Row of an Hadamard Matrix

While conceptually simple it's easy to notice how rapidly the method described in section 3.1.1 becomes computationally intensive with the rise of k. With this in mind and taking advantage of the high symmetry level of the matrices that result of SC, the relationship between the elements of these matrices were studied aiming at the development of an algorithm that would allow the construction of isolated rows of an SC Hadamard matrix. The following algorithm is the result of the aforementioned considerations and with it one can generate a Hadamard matrix's row while simply using as input that row's index and the rank of the matrix.

Let us consider the matrix $H_n : n = 2^k \wedge k \in \mathbb{N}$, with elements $h_{ij} : i, j = 0, 1, ..., n-1$, then the relationship expressed in eq. 3.4 is verified.

$$h_{i(\frac{n}{2}+r)} = \begin{cases} h_{ir}, & i = 0, ..., n/2-1 \\ -h_{ir}, & i = n/2, ..., n-1 \end{cases} \quad , \quad \forall r \in \{0, ..., n/2-1\} \tag{3.4}$$

Every element of the second half of the ith row are then equal to or the symmetric of the corresponding elements of the first half. We can then consider only the first half, and the block matrix $H_{\frac{n}{2}}$ in which it is included, do the same procidure and verify the same property; and repeat up until the sectioned row considered only has one element.

From the SC method results that $h_{i0} = 1, \ \forall i \in 0, ..., n-1$, which shall then be the starting point for the construction of any row. To determine the remaining elements of a row some tests will be realized on the rows index i to identify the block matrices to which the progressively bigger sectioned rows belong. The pseudo-code for the algorithm is presented below:

1. Define $h_{i0} = 1$ and $k = log_2(n)$;
2. Set $l = 0$;
3. Define $P_l = (i \div 2^l) \mod 2$, where $(\div)$ stands for the integer division;
4. $h_{i(r+2^l)} = \begin{cases} h_{ir}, & P_l = 0 \\ -h_{ir}, & P_l = 1 \end{cases} : \quad r \in \{0, ..., 2^{l-1}-1\}$;
5. Increment l one unit;
6. Repeat from 3 to 5 while $l < k$.

3.3.2 Libraries

Given the repetitive nature of the instructions that are required to establish minimal communication with the DM365, most of these were created and defined within libraries as to visually lighten the control program main structure (see appendix C.2). These functions were sorted to three libraries according to their purpose.

The ones directly involved with the manipulation of packets were placed in `"PKT.h"`; those that deal with pattern array generation and manipulation so that the patterns to be loaded into DM365 will not only control the behavior of the desired set of micromirrors of the DMD as will be in conformity with the `BMP` file format, were placed in `"BMP.h"`; at last, auxiliary function unrelated to either of those subjects were placed in `"SHELF.h"`.

PKT.h

The packets traded between PC-DM365 are 8 bit arrays, as such these are declared as being of the type `unsigned char`. To skim the nomenclature that type was here renamed as `int8` using a `typedef` declaration; for the same reason and in the same way the `unsigned short` was renamed `int16`.

Some recurring values used for packet communication (packet type, flags, trigger types, among others) were here globally declared using `enum` declarations.

The functions here defined are responsible for assembling packets and sending/receiving them and are based on the functions present in the LightCrafter's GUI's build files.

- `void Fill_PKT_Header(int8 type, int16 command, int8 flag, int16 data_Length, int8* packet)`

  This function fills the elements of an array pointed to by `packet` with the variables `type`, `flag`, `command` and `data_length`. Its purpose is to effectively define the packet's header.

- `int8 calcChecksum(unsigned int N, int8* packet)`

  Uses `N` elements of an array pointed to by `packet` to return a value calculated using eq. 3.5, this value is then placed in the $N+1$th position of that array. In packet communication `N` should be the dimension of the packet's load plus its header, with the purpose of this function being to provide and store a control byte for the packet.

$$B_N = \left(\sum_{i=0}^{N-1} B_i \right) \bmod (0 \times 100) \tag{3.5}$$

- `void PKT_Seq_Set(int8 numberPat, int8 trigger, int8 led, int8* packet)`

  This function was written to easily edit the settings of a pattern sequence. To do this the function will fill the elements of an array pointed to by `packet`, which are analogous to a packet's load, with the variables present in its signature and default ('0' or '1') values on the options that it are not relevant or in our interest to edit. The type of packet which load is supposed to be assembled here possesses a 17 byte load that has the following structure (the numbering starts at 6 for, as mentioned in section 3.2.2, the header is carried in bytes 0 through 5):

  - B_6 - pattern bit depth, can assume values '1' through '8' and will be here set to '1' which is associated to monochromatic patterns.
  - B_7 - number of patterns to be included in the sequence, can assume values '1' through '96' and will be here set to the value of `numPat`.
  - B_8 - signals whether the inverse patterns of those loaded into the sequence should be inclueded in it, can assume the values '0' or '1' (for 'no' and 'yes', respectively) and is here set to '0'.
  - B_9 - trigger type to be used to advance the sequence, can assume values '0' through '6' and will be here set to the value of `trigger` (which in this application will always be set by the global variable `EXTERNAL_POS` (TTL rise) to which corresponds the value '2').

- B_{10} ... B_{13} - LSB ... MSB of the delay applied to the trigger, in microseconds, will be here set to '0'.
- B_{14} ... B_{17} - LSB ... MSB of the trigger period, in microseconds, should the trigger type chosen be defined as automatic, as such will be here set to '0'.
- B_{18} ... B_{21} - LSB ... MSB of the exposition time, in microseconds, will be here set to '0' indicating that the exposition time shall be the whole time interval between triggers.
- B_{22} - indicates which led is to be lit and is here set to the value of `led` (even though the LightCrafter's leds have been removed in this application this is data is still required to be set).

- `int pkt_say(int socket, int8* packet, unsigned int pkt_length)`

  This function sends an array, pointed to by `packet`, of size `pkt_length` through a port which value is given by `socket`. Its purpose is to send a packet to the DM365 without expecting a response. The function returns '0' if successful or '-1' and an error message should it not be capable of sending the whole packet.

- `int pkt_talk(int socket, int8* packet, unsigned int pkt_length, int16 data_length)`

  As the previously described function, `pkt_talk()`, sends a packet of length `pkt_length` to the DM365; then it checks the header for the type of the packet it just sent in order to determine the type of packet it expects to receive in response; the function continues by reading the header of the response packet and comparing its type against the type it expected; then it stores the response packet's dataload size in the variable `data_length`; at last the function reads the load of the packet and checks its integrity by comparing its checksum byte with the result of parsing that packet throught the `calcChecksum()` function. The function returns '0' if successful or '-1' and an error message should it not be capable of sending the whole packet or the size of data received or packet type are different of what was expected, with an error message specifying the case.

The detailed code of this library's functions can be consulted in the appendix C.3.

BMP.h

In order to change the state of each of the DMD's micromirrors it's necessary to load and display a pattern that should be in BMP file format (this format's structure shall be addressed when describing the `Fill_BMP_Array()` function). This libraries functions are in this way responsible for shaping a matrix row's elements into data that can be interpreted by the LightCrafter as being a pattern.

These functions will be used to generate rows of an H_{2^k} matrix using the algorithm, described in section 3.3.1, these rows' elements will be used to create square matrices of rank=$2^{k/2}$ which will then be adjusted to be accurately displayed, centered on the DMD's grid and fitted into the BMP format.

- `void Create_Hadamard_Lines(int index, int k, int8* line, int* pots)`

Generates and stores the elements of the row indexed by the number `index` of an Hadamard matrix, of rank=2^k, that could have been calculated with SC, using the algorithm described in section 3.3.1. Additionally, those elements that should have value '-1' are filled with '0' since in the data processing done by the LightCrafter, in the context of micromirror state, only the values '1' and '0' have meaning (+12°and -12°, respectively).

- `void Expand_Hadamard_Map(char* OrigLine, char* DefLine, int OrigSize)`

  This function was written should there be a need to use patterns that are generated from a k smaller than the maximum allowed. In that event to keep the representation of that pattern from using even less mirrors, this function will receive the pattern's elements, stored in `OrigLine`, and fill the elements of `DefLine` such that to each of the first array's elements corresponds a power of 2 number of elements as to make the total number of elements in `DefLine` equal to 256×256 (the biggest base 2 square that can be represented in a 608×684 diamond grid as per eq. 3.3). The number of elements used in the second array to match to elements of the first is determined using the proportion between number 256×256 and `OrigSize`.

- `void Transform_Map(int8* Line, int8* HadLine)`

  The purpose of this function is to manipulate the relative positions of the elements of `HadLine`, with that being analog to coordinate transformations corresponding to a centering and a 45°rotation of the pattern on the diamond grid, and store these new positions on `Line`. The impact of this function can be better grasped with the help of the example that follows and fig. 3.4.

  > **Example:** Let us consider a 16 element binary row [1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1] that can be used to generate a 4×4 pixels pattern; let us also consider a 7×11 diamond grid. In 3.4a the coordinate system for a DMD-like diamond grid; in 3.4b is the result of loading the row without manipulating its elements; in 3.4c is the result of accounting for the mirrors that aren't to be used; at last, in 3.4d the result of centering and coordinate system shifting is displayed as an accurate representation of the pattern despite its rotation.

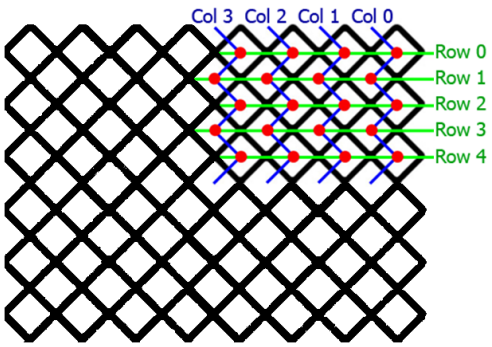
- `int Fill_BMP_Array(int8* BMP_array, int8* pixel_map_array, int BMP_length, int BMP_payload_length)`

  This function expects an array, `pixel_map_array`, containing one pixel value per element, to generate a byte array, `BMP_array`, compliant with the BMP file format and which payload will hold 8 pixel values per element (conformed with the monochromatic BMP structure which minimizes the file size)[27].

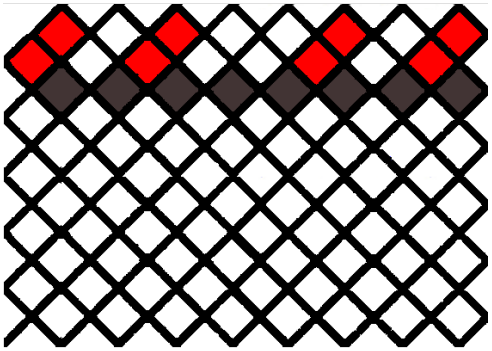
- `int Write_BMP_File(string bmpName, int val_length, double values[])`

  This function takes as inputs a string `bmpName` which will be used to name the image file to be created, an array of values to be translated into a grey scale `values[]`, and the length of that array `val_length`. With these the function will create a BMP file in a 256 tones grey scale.

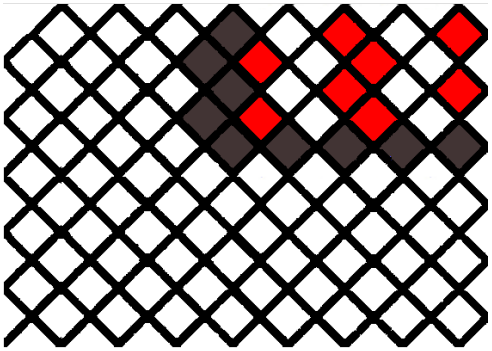
The detailed code of this library's functions can be consulted in the appendix C.4.



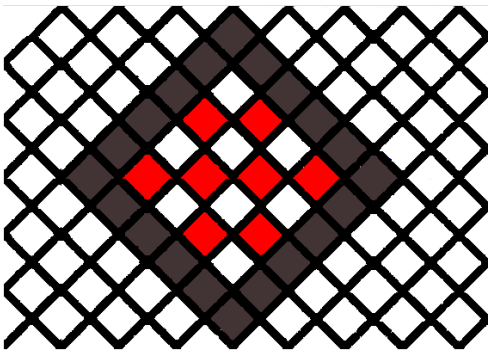
(a) *DMD micromirror grid indexation.*



(b) *Representation of the pattern on the grid without manipulation of its elements.*



(c) *Representation of the pattern after applying offsets to its elements but not accounting for the distortion due to the mismatch between a pattern with elements with orthogonal indexation in a diamond grid.*



(d) *Representation of the pattern after transforming the coordinates of its elements to fit and center them on the diamond grid.*

Figure 3.4: *Example of pixel indexation, for the pattern that's generated by the row [1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1], in a diamond grid resembling that of the DMD. In this representation to the value '1' corresponds the color red and to '0' corresponds white, while in dark gray is simply a separation of '0's from the original row and added '0's to control the micromirrors uncounted for by the pattern.*

SHELF.h

- `void Random_Lines(double seed, int k, int *Lines, int max)`

  This function was created to generate and store numbers to be used as row indexes of a matrix. It uses the `rand()` function and some verification instructions to generate `k` non-repeating pseudo-random integers between 0 and `max`, which are then stored in an array pointed to by `Lines`.

- `int Binary_Choice(string query)`

  Binary option menu-like function which prints the string `query` to the console and waits for the user's input, that should be either 0 or 1, and returns that value.

The detailed code of this library's functions can be consulted in the appendix C.5.

3.3.3 DMD-CS.cpp

The purpose of the set of instructions in this program is to provide a simple console-based interface between the DM365 and the user in order to easily manipulate the DMD. The user can chose one of two USB ports (the options of this segment should be edited to fit the PC in which the program is running); the size of the patterns to be used (which is, however, limited by the constraints of SC); and number of patterns to generate and be placed in a sequence. A flow chart representative of the processes to be described next can be consulted in the appendix C.1.

To establish communication with the DM365 a `socket` structure is defined using default values (which were taken from the LightCrafter's GUI build files). The program attempts to establish communication with the DM365 and, if it is successful, a 1 s hiatus follows (recommended by the manufacturer) upon which a packet to read the LightCrafter's software and firmware versions is constructed and sent to the DM365. A response packet is received containing the requested data which is then printed to the console. Afterwards the user is asked about which USB port to use and, once the choice is made, a Serial port structure is defined using the `RS232` library[49]. The Serial port is opened and the program sends a message through it to prompt the acquisition subsystem's controller to start running its program (this will be described in section 6.1.1). The user is prompted to define the integration time for the each measurement and the size of the patterns that are to be generated by choosing k (this choice being constrained to $4 < k < 16 \wedge k \bmod 2 = 0$. The program waits to receive the name of the file, through the Serial port, which will be saved in the SD Card, and then prints it to the console to inform the user.

The LightCrafter is set to pattern sequence display mode and the user is requested to provide the number of patterns, `pat_Number`, he wishes to generate. That number of non-repeating random row indexes is generated and stored in an array, `line_Numbers`, and is also wrote to a file (which is stored in the same folder as the program's execution file) with the same name as the measurements file but prefixed with an "I"; that file's name is then printed to the console. The number of pattern sequences, `seq_Number`, necessary to display all patterns is calculated.

A loop starts to iterate over these sequences up until `seq_Number` is reached. In it a command to edit the pattern sequence settings is sent specifying the number of patterns to include in the sequence, `seq_Lim`, and the type of trigger to use; these settings will only be edited again if more than one pattern sequence is required to display all patterns and before loading the last batch of patterns, all other sequences will be loaded with the maximum number of patterns, `seq_Max`.

> Another, inner, loop starts to iterate over the number of patterns to generate for the present sequence until it reaches `seq_Lim`. Using an index stored in `line_Numbers` an Hadamard matrix row is created, using the algorithm described in section 3.3.1, and stored in an array `had_Line`; if the chosen k is the maximum allowed value, `had_Line` is used as input to the `Transform_Map()` function, if not that array will be used as input to the `Expand_Hadamard_Map()` function and the output, the array `exp_had_line` will be the input `Transform_Map()`.
>
> The output of `Transform_Map()`, the array `transf_had_line`, is inputed into the `Fill_BMP_Array()` function which outputs the data that's then loaded in a packet which is sent to load the pattern into the sequence.

Once all `seq_Lim` patterns are generated and loaded into the present sequence, a command is sent to start its display. Then, using the `RS232_SendByte()` function, `seq_Lim` is sent to

the Arduino which will proceed to control the acquisition processes while triggering the pattern advancement in the sequence. After all patterns in the sequence have been displayed the Arduino writes back through the Serial port while on the other end the function `RS232_PollComport()` is being used to check for the Arduino's answer while waiting to proceed. At this point a command to stop the sequence display is sent.

Once all sequences are displayed, and the loop is over the user is prompted to decide whether to restart or to terminate the program.

For more details consult the code present in appendix C.2.

3.3.4 Testing

The program was tested using the DLP LightCrafter with its optics and leds still attached, in order to visually verify that the DMD was being manipulated as desired through the display of the generated patterns. While testing, when asking for 10,000 patterns to be displayed, divided in sequences of 96, the average time of transmission for each 96 patterns through the PC-DM365 connection was determined at approximately 6s. This is clearly an issue, since the target sampling rate is in the 2 kHz to 4 kHz range.

After contacting the manufacturer we learned that even though the technical specifications of the LightCrafter indicate that it has a USB 2.0 port, as a safety measure the implemented protocol is actually USB 1.1 to which corresponds a max transfer rate of 1.5 MB/s. Considering that after going through the `Fill_BMP_Array()` function the generated pattern results in 52 kB of data, and that a sequence will then yield approximately 5 MB, the transfer rate is calculated at ≈ 830 MB/s which is within the range of the expected speed for USB 1.1. Another source of delay was found later, and will be presented in the next section.

3.4 Testing DLP's Pattern Switch Rate

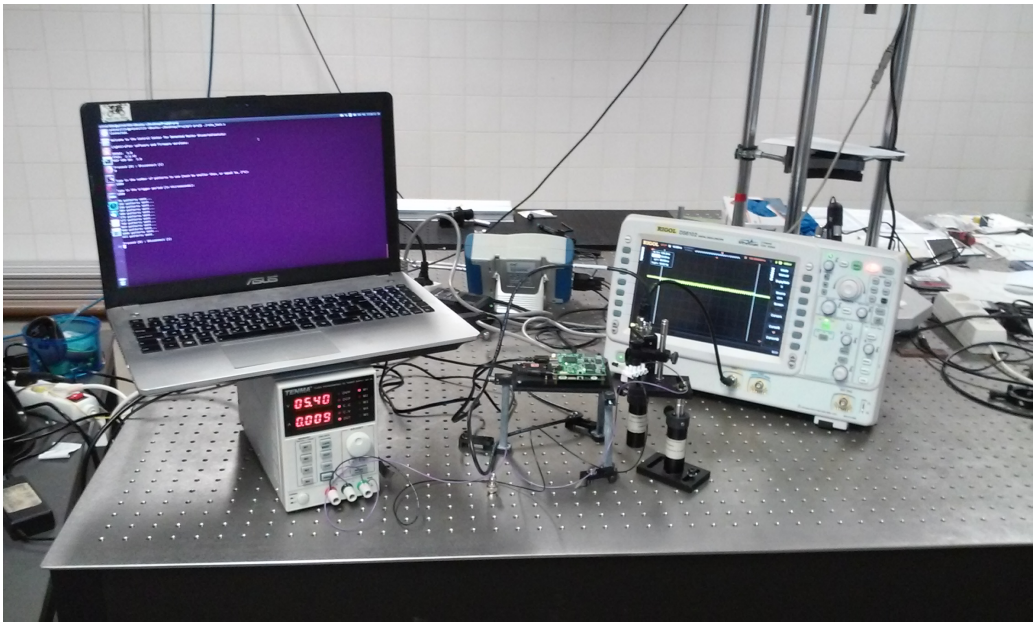


Figure 3.5: *Test setting for the LightCrafter's monochromatic pattern sequence switch rate.*

Even though it is not currently possible to achieve an effective sampling rate in the [2;4] kHz

range for a number of patterns that exceeds the regular sequence limit, 96, due to the data transfer speed limit, a procedure was designed and performed to verify whether or not that range was achievable by the DMD when using just was one sequence.

The following equipment was used to perform this procedure and is represented in fig. 3.5:

- 2.2 kΩ resistor;
- 50 Ω BNC cable;
- 950 nm led;
- DLP LightCrafter without optics not leds;
- IR target;
- Oscilloscope Rigol DS6102[50];
- PC running an adapted version of `DMD-CS.cpp`;
- Photodiode PDA 100a[51];
- Power source TENMA 72-2535[52].

The power source was connected to the led which was connected on its other end to the resistor which then connected to the ground provided; the BNC cable was used to connect the output of the photodiode to input channel 1 of the oscilloscope.

Using the oscilloscope to measure the output from the led and using the LightCrafter GUI to force all mirrors to tilt to one side, the led and the photodiode were positioned in front of the DMD such as to maximize the photodiode's output (see fig. 3.6).

For this procedure instead of using the `Create_Hadamard_Lines()` function an alternative version which generated just two arrays, all 0's or all 1's. This way the DMD will display in its center a blinking square of tilting mirror and we're able to measure the blinking rate. 48 maximums and 48 minimums are expected to be observed. More so, the version of `DMD-CS.cpp` used was independent of the Arduino and acquisition, and us such it sends triggers via packets.

At first very low frequencies, from 1 to 33 Hz, were tested with successful results. On the second run, using frequencies from 50 to 400 Hz, the results were unexpected for even though the frequencies measured appeared to be approximately the ones we were supposed to measure, the number of peaks observed decreased with the increase of the frequency.

Postulating that maybe the DM365 wasn't being able to process all the given instructions before receiving the first trigger to display a pattern, a 1 s sleep time was added to the control program after it sends the `START` command to the DM365. This allowed to observe the expected results for frequencies up to 3 kHz. Later it was verified that a minimum of 350 ms of sleep time was required.

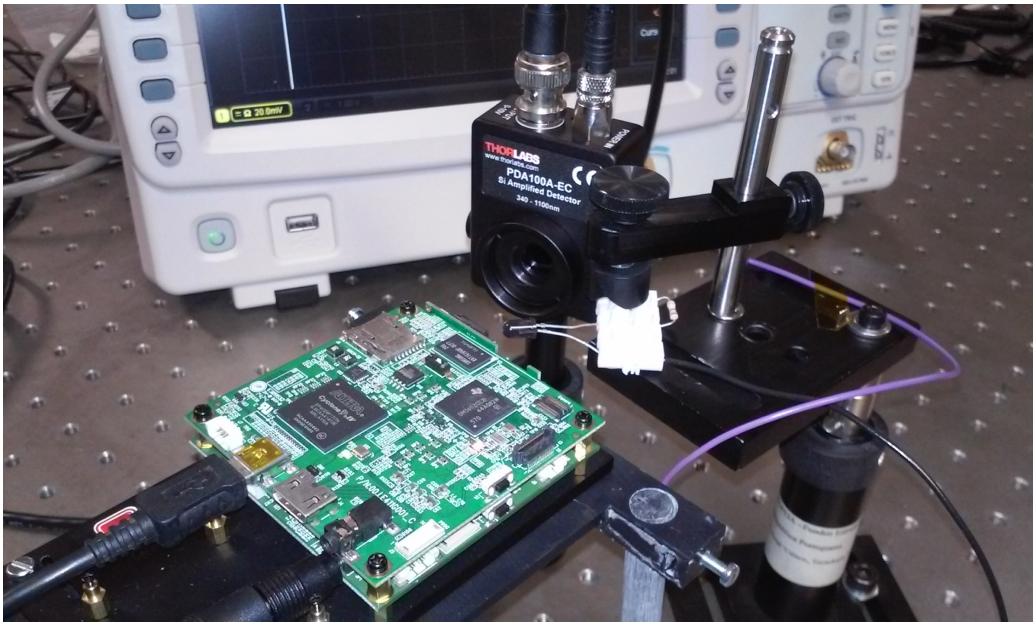


Figure 3.6: *Display of the relative positions of the led, LightCrafter's DMD and the PDA.*

Chapter 4

Development of a Signal Reconstruction Subsystem

Once a light signal has been processed by the acquisition subsystem we're left with a vector which, assuming the signal is sparse on the coding basis, holds the projections of that signal on a small set of vectors of that basis. The only thing left then to obtain that signal is recover its coefficients on the used basis and reconstruct it.

The algebraic formulation of the problem is quite simple but it involves matrix inversion, a computationally very demanding task and which execution, for large matrices, is too time/power consuming. In cases such as this numerical methods can be employed instead, for even though the approximated solutions yielded by these methods are dependent on the number of times the algorithm is iterated, unless an unreasonable stop conditions are inputted into the system, such algorithms are bound to be less computer-intensive.

In this chapter the implementation of one such algorithm is described.

4.1 Description

Primal-dual interior-point methods are a class of algorithms used to solve linear and non-linear convex optimization problems. One such method has been described, by Boyd and Vandenberghe, as a linear programming algorithm, in Chapter 11 of [53] which has been successfully implemented as a Matlab script, by Candès and Romberg[54], to solve the Basis Pursuit problem[55], or ℓ_1-minimization linear program (this, and other scripts employing different optimization methods, such as conjugate gradients (CG), or algorithms, like total variation minimization (TV), were released as the `l1magic-1.11` package[26]). Alternative Matlab scripts were later developed and presented, such as Koh, Kim and Boyd's ℓ_1-regularized least squares[56], or Figueiredo, Nowak and Wright's gradient projection[57] (GP), which can be found in the `GPSR 6.0` package[58].

An outline of the primal-dual algorithm developed by Boyd and Vandenberghe follows, as well as comparative results of applying the different scripts that can be found in the previously mentioned packages.

4.1.1 A Primal-Dual Interior-Point Algorithm

The algorithm presented in [53] is a standard-form linear program

$$\min_z \langle c_0, x\rangle \quad s.t. \quad A_0 x = y,$$
$$f_i(x) \leq 0,$$

where A_0 is a $K \times N$ matrix, $x \in \mathbb{R}^N$, $b \in \mathbb{R}^K$ and f_i, $i = 1, ..., m$ are linear functionals of the form

$$f_i(x) = \langle c_i, x\rangle + d_i$$

where $c_i \in \mathbb{R}^N$, $d_i \in \mathbb{R}$. Applying the Karush-Kuhn-Tucker conditions eq. 4.1 to the program, the algorithm will find the optimal $x^\star$ along with optimal dual vector $\nu^\star$ and $\lambda^\star$ to satisfy them, by solving that set of non-linear equations such that $\nu^\star \in \mathbb{R}^K, \lambda^\star \in \mathbb{R}^m \geq 0$.

$$\begin{aligned} c_0 + A_0^T \nu^\star + \sum_i \lambda_i^\star c_i = 0, \\ \lambda_i^\star f_i(x^\star) = 0,\ i = 1, ..., m, \\ A_0 x^\star = y, \\ f_i(x^\star) \leq 0,\ i = 1, ..., m. \end{aligned} \tag{4.1}$$

To reach a solution the Newton method is employed at an interior point (x^l, ν^l, λ^l) where the system is linearized and solved, with the step to the new point $(x^{l+1}, \nu^{l+1}, \lambda^{l+1})$ modified so as to stay in the interior. This is achieved by biasing the solutions of the linearized equations towards the interior, that is by relaxing the condition $\lambda_i f_i = 0$ to

$$\lambda_i^l f_i(x^l) = \frac{1}{\tau^l}, \tag{4.2}$$

and increasing τ^l at each Newton iteration[59, 60].

To quantify how close a point (x, ν, λ) is of satisfying eq. 4.1, with eq. 4.2, one calculates the primal, dual and central residuals

$$\begin{aligned} r_{dual} &= c_0 + A_0^T \nu + \sum_i \lambda_i c_i \\ r_{cent} &= -\Lambda f - \frac{1}{\tau}\mathbf{1} \\ r_{pri} &= A_0 x - y \end{aligned}$$

where Λ is a diagonal matrix with $\Lambda_{ii} = \lambda_i$, and f is vector which components are f_i.

Next, the step $(\Delta x, \Delta\nu, \Delta\lambda)$ such that

$$r_\tau(x + \Delta x, \nu + \Delta\nu, \lambda + \Delta\lambda) = 0,$$

must be found. Linearizing through Taylor expansion around (x, ν, λ),

$$r_\tau(x + \Delta x, \nu + \Delta\nu, \lambda + \Delta\lambda) \approx r_\tau(x, \nu, \lambda) + J_{r_\tau}(x, \nu, \lambda) \begin{pmatrix} \Delta x \\ \Delta\nu \\ \Delta\lambda \end{pmatrix},$$

where $J_{r_\tau}(x, \nu, \lambda)$ is the Jacobian of r_τ, we get

$$\begin{pmatrix} \mathbf{0} & A_0^T & C^T \\ -\Lambda C & \mathbf{0} & -F \\ A_0 & \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta \nu \\ \Delta \lambda \end{pmatrix} = - \begin{pmatrix} c_0 + A_0^T \nu + \sum_i \lambda_i c_i \\ -\Lambda f - \frac{1}{\tau}\mathbf{1} \\ A_0 x - y \end{pmatrix}$$

with C being an $m \times N$ matrix which has c_i as rows, and F a diagonal matrix with $F_{ii} = f_i(x)$. Eliminating $\Delta\lambda$ the system is simplified to

$$\begin{pmatrix} -C^T F^{-1} \Lambda C & A_0^T \\ A_0 & \mathbf{0} \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta \nu \end{pmatrix} = \begin{pmatrix} -c_0 + \frac{1}{\tau} C^T f^{-1} - A_0^T \nu \\ y - A_0 x \end{pmatrix} \tag{4.3}$$

Having determined the step direction $(\Delta x, \Delta \nu, \Delta \lambda)$, it's then necessary to determine its length $0 \leq s \leq 1$ and for this two criteria must be satisfied:

1. $\forall i \; f_i(x + \Delta x) < 0 \wedge \lambda_i > 0$, which means that both x^l and λ^l must remain in the interior for all l.

2. The euclidean norm of the residuals has decreased sufficiently:

$$\| \, r_\tau(x + s\Delta x, \nu + s\Delta\nu, \lambda + s\Delta\lambda) \, \|_2 \leq (1 - \alpha s) \cdot \| \, r_\tau(x, \nu, \lambda) \, \|_2,$$

   where α is a user-specified parameter (here set as $\alpha = 0.01$).

Setting $s = s_{max}$, where s_{max} is the maximum value of s that satisfies criterion 1, criterion 2 is checked and, if not satisfied then s is reduced by a factor β (here set as $\beta = 0.5$).

At last, if r_{dual} and r_{pri} are small, the surrogate duality gap $\eta = -f^T \lambda$ is an approximation to close a certain point (x, ν, λ) is to being optimal (i.e $\langle c_0, x \rangle - \langle c_0, x^\star \rangle \approx \eta$). The algorithm repeats this altered version of Newton iterations until η falls below a user-specified tolerance.

A more detailed exposition can be found in [53, 54, 60].

In this algorithm, most of the computational effort is spend on solving eq. 4.3. In the present application, where N and K are large, not only isn't $-C^T F^{-1} \Lambda C$ easily invertible as there are equality constraints, which renders solving the set of linear equations of eq. 4.3 infeasible. Instead, "matrix free", iterative solvers, such as Conjugate Gradients (CG) or Bi Conjugate Gradient Stabilized (BiCGSTAB), are used to attain a fast and accurate solution. Such solvers instead of loading entire matrices into the memory and compute the operations, use instead index references and function handles to retrieve said matrices items from those indices.

4.1.2 Comparing Scripts

In order to chose which solver to implement in C/C++ some tests were performed. The following scripts were tested:

- From the `l1magic-1.11` package:
  - `l1eq_pd.m`
  - `tvqc_logbarrier_pd.m`
- From the `GPSR 6.0` package:
  - `l1_ls.m`

– `GPSR_BB.m`

All of the above, except for `tvqc_logbarrier.m` which uses the Symmetric LQ method (SYMMLQ), use the CG method to compute eq. 4.3, varying only in the type of constraints applied to the problem and/or how the instructions are implemented to maximize their efficiency. In [60], `l1magic-1.11` scripts, `l1_ls.m` and `GPSR_BB.m` are compared in regards to processing time, with `GPSR_BB.m` coming out on top by presenting better scalability with the size of the problem. In the present application however, processing time comes second to denoising capability.

ℓ_1-norm minimization, TV and GP algorithms try to achieve the same goal through different routes. Because most times these iterative solvers won't find one exact but one of several approximate solutions, costraining this set via ℓ_1-norm minimization increases the probability of finding a sparse vector as a solution; TV regularization is based on the principle that signals may have excessive detail and thus present high total variation, and will instead try to minimize that total variation while attempting to keep the result as close match of the original, removing unwanted details but keeping important features such as edges; GP uses a different approach to solve the ℓ_1-norm problem, using special line search and termination techniques to converge to a solution faster than interior-point techniques[58], in the case of the `GPSR_BB.m` script, in addition to the equality constraints applied to the ℓ_1-norm term, another term is considered (a noise term) to which quadratic constraints are applied.

In order to compare how each script performs, a test image was picked (see fig. 4.1) and simulations of CS acquisitions were performed, each outputting a sampling matrix file and a measurements file for the scripts to parse.

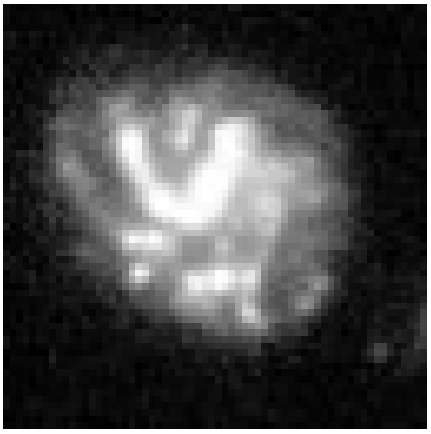
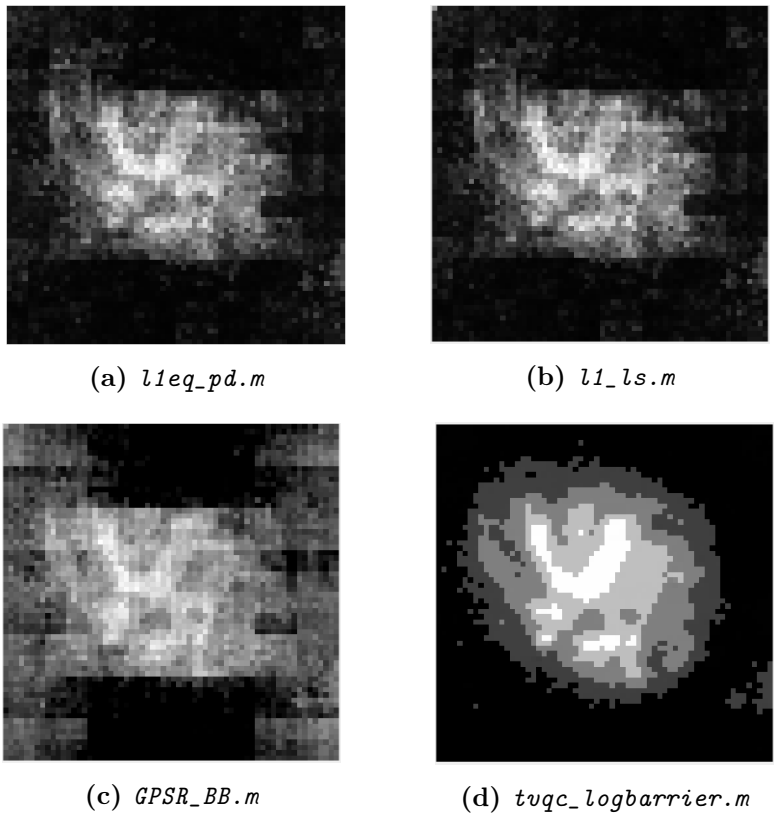


Figure 4.1: *Image of a galaxy. This image was used to generate simulations of measurements performed using the CS technique.*

These simulations were performed using a matlab script written by Krone-Martins and Bandarra to emulate the acquisition processes of a CS camera. In it: a sampling matrix is first generated; then a source image's pixel value levels are converted into a scaled number of photons; the 'measurement' is then performed by using the vectors of the sampling matrix to filter which 'photons' are detected; this signal is integrated and multiplied by a gain factor (optionally Poisson noise can be applied at this stage); when required, gaussian noise can be added to each observation; finally, both sampling matrix and observation vector are saved to

files. All of the simulations performed in the following tests were built using an Hadamard matrix of rank 4096 (resulting in 64×64 patterns). The code is presented in appendix C.13.

Fig. 4.2 presents the reconstructions of a simulation, created using fig. 4.1 as a template, with a low 'photon' flux (a maximum of 5 'photons' per pixel) and no noise. As it can be seen, most features were in this way successfully reconstructed, with `tvqc_logbarrier.m` presenting these more clearly. More of these preliminary tests were made, using the same parameters but different vectors of the sampling matrix; unfortunately though, `l1eq_pd.m` and `l1_ls.m` were not capable of successfully reconstruct most of them due to, during that process, the handled objects no longer having the necessary properties thus triggering errors. As such, these two algorithms were discarded and the next remaining batch of tests were only performed on `GPSR_BB.m` and `tvqc_logbarrier.m`.



(a) *l1eq_pd.m* **(b)** *l1_ls.m*

(c) *GPSR_BB.m* **(d)** *tvqc_logbarrier.m*

Figure 4.2: *Reconstructions of fig. 4.1, from simulated CS measurements, using four distinct Matlab scripts.*

Next, the reconstruction algorithms were tested for their accuracy's dependence on the number of measurements performed. For this, the simulation script was complemented to generate multiple sampling matrices and observation vectors with each new matrix and vector having twice as many elements while keeping the elements of the previous ones.

To measure the algorithms' performance the source image pixel values were normalized and saved; then, for each normalized element of each reconstruction, the relative error was computed acording to eq. 4.4; finally, a statiscal analysis of the computed data was performed. First the average and standard deviation were calculated, but for some reconstructions these values resulted in +Inf, while in most of these even if not infinity the average was nevertheless much greater than expected; this illicited te suspicion that a very small set of data elements were skewing the results, as such is was decided to calculate the median and median absolute deviation

(MAD). Plotting histograms for those data sets and comparing them with the aforementioned parameters it became clear that the median and MAD better characterize the distribution of those sets (see fig. 4.3).

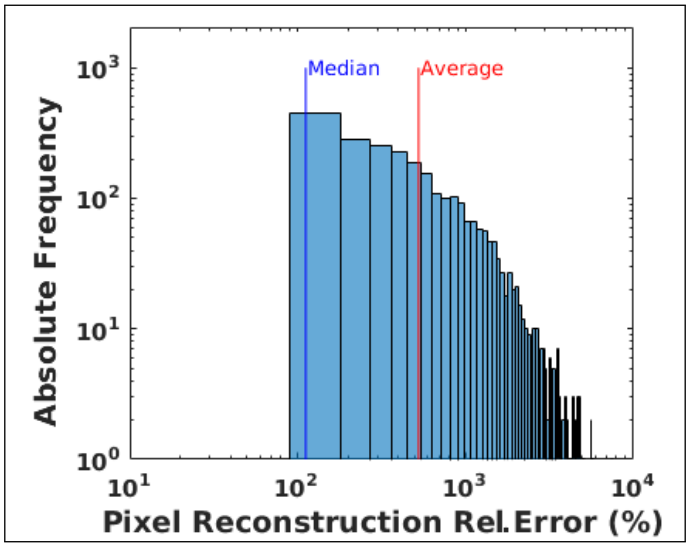


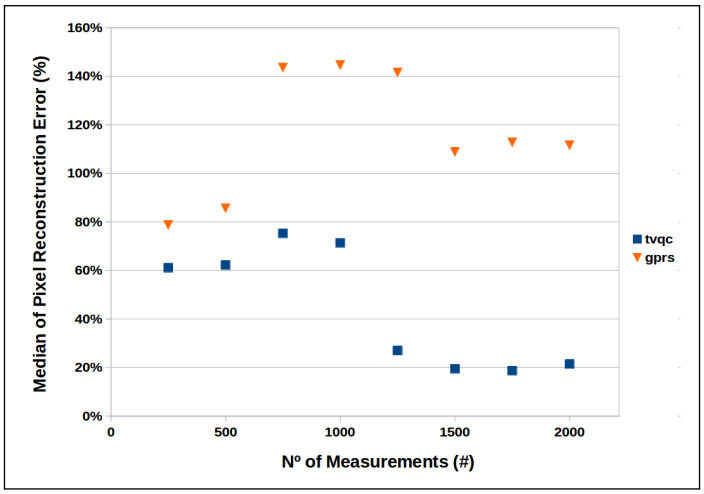
Figure 4.3: *Histogram presenting the absolute frequency of a reconstruction's pixels relative error.*

In fig. 4.4 the results of this test are plotted, and there we can see that `tvqc_logbarrier.m` displays overall better results. Additionally, even though from the graph it appears that for a reduced number of measurements both scripts' performance is similar, looking into the actual images (see fig. 4.5) this appears to be simply a mathematical effect probably due to the chosen evaluation method, since the `GPSR_BB.m`'s results aren't visualy recognizable, unlike `tvqc_logbarrier.m`'s.

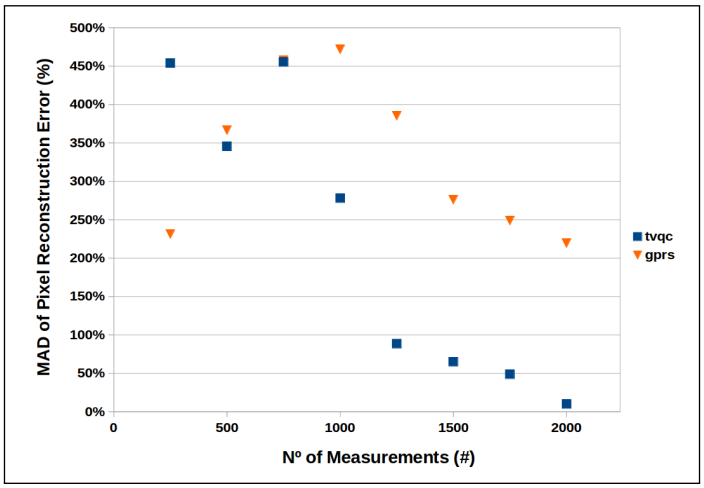
$$\varepsilon(\%) = \frac{|x - x^\star|}{x} \times 100\% \tag{4.4}$$

Having estimated that a relative error (median + MAD) inferior to 100% occurs at approximately 1500 measurements (which is roughly 3/8 of the available vectors in the basis used for this simulation) and having visually confirmed that the reconstructed image was recognizable (see fig. 4.6), that number was set for the next simulation batch.

The simulation script is prepared for simulating both Poisson noise (which intends to emulate uncertainty inherent to a flux of photons hitting a detector) and gaussian noise (to emulate all noise due to electronics). The Poisson noise impact can be mitigated by increasing the photon count, which can be done by increasing either the number of photons per pixel or the integration time; as such, for the following simulated measurements, the 'photon' flux was increased 200× in relation to the initial simulations, and Poisson noise was also simulated. Since the average value of gaussian noise will only offset the signal, it is of no consequence for this application, as such in the last set of simulated observations, gaussian noise centered at 0 was added. The gaussian noise impact however will be dependent on the relation between the wideness of its distribution and the average of the observation vector's elements; a metric akin to usually employed signal to nosie ratio was then constructed, a variance to signal ratio (VSR). All of these observations used the same measurement vectors, but each new one the variance of the gaussian noise was doubled.



(a)



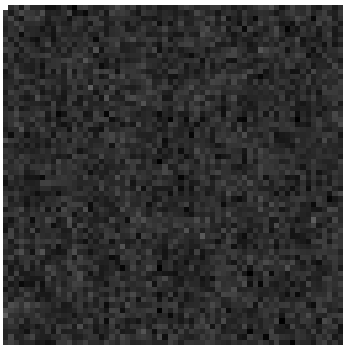
(b)

Figure 4.4: *Median (fig. 4.4a) and MAD (fig. 4.4b) of a reconstruction's pixel relative error plotted against the number of measurements used in that reconstruction.*

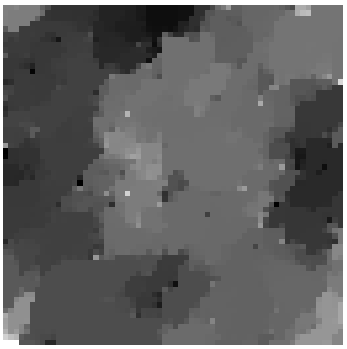
Once again, the median and MAD of the reconstructions' pixel relative error was computed[1]. The results were plotted against the gaussian noise VSR and can be consulted in fig. 4.7. Despite some fluctuations it is clear that both the median and MAD increase with the VSR; relative errors (median + MAD) inferior to 100% occur for VSR<10% but, nevertheless, the reconstruction only starts to lose feature clarity for VSR≈2000% (see fig. 4.8) which corresponds to a maximum relative error (median+MAD) of ∼160%.

Considering the satisfactory performance of `tvqc_logbarrier.m` even for VSR>1000%, and the limitations of the remaining scripts, this script was chosen to be implemented as the signal reconstruction subsystem.

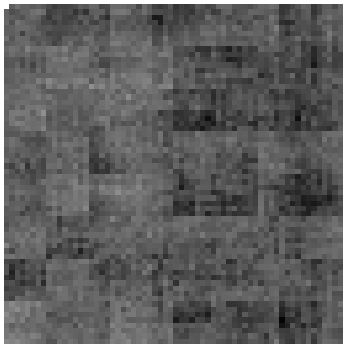
[1]Unfortunatly `GPSR_BB.m`'s displayed behaviour with the increase of 'photon' flux came out short of what was expected and, as such, its performance wasn't tested against gaussian noise VSR.



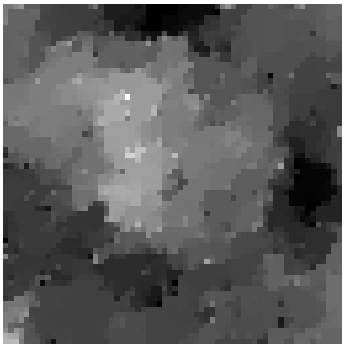
(a) *N=250,* `GPSR_BB.m`.



(b) *N=250,* `tvqc_logbarrier.m`.



(c) *N=500,* `GPSR_BB.m`.



(d) *N=500,* `tvqc_logbarrier.m`.

Figure 4.5: *Reconstructions of fig. 4.1 for simulated observations where with a maximum 'photon' flux of 5 per pixel, no noise, and N measurements.*

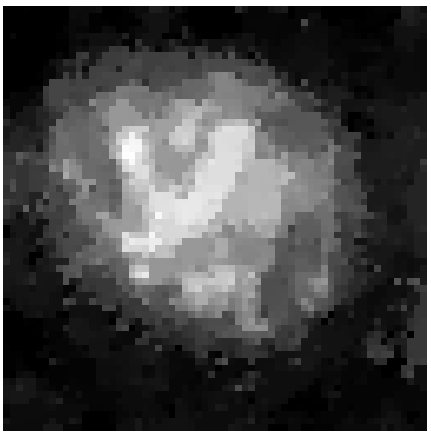
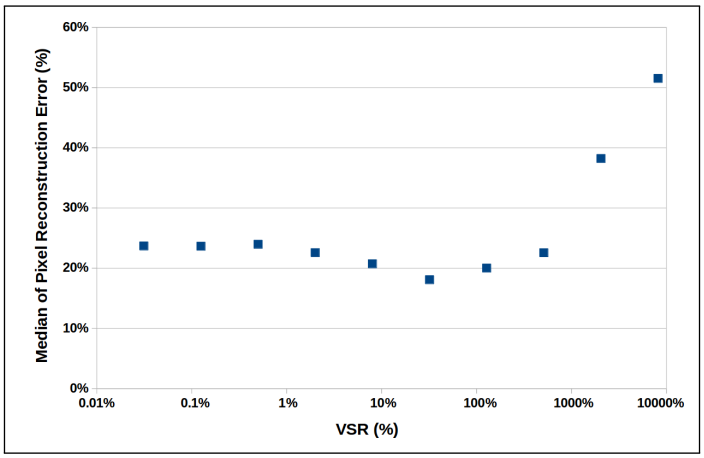
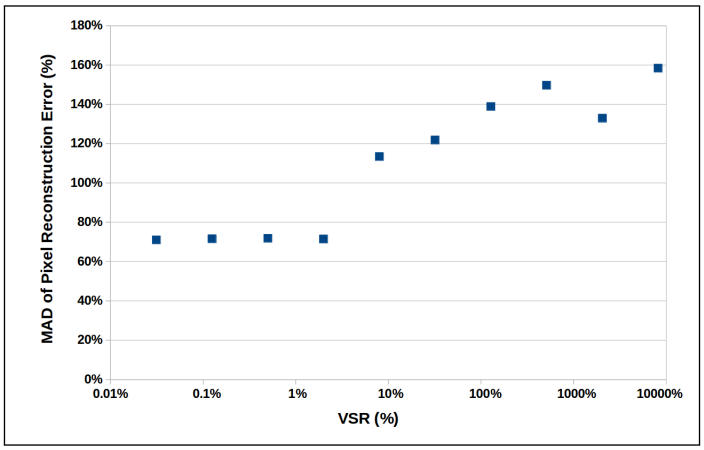


Figure 4.6: *Reconstruction of fig. 4.1, using 1500 simulated measurements without noise and the* `tvqc_logbarrier.m` *script.*

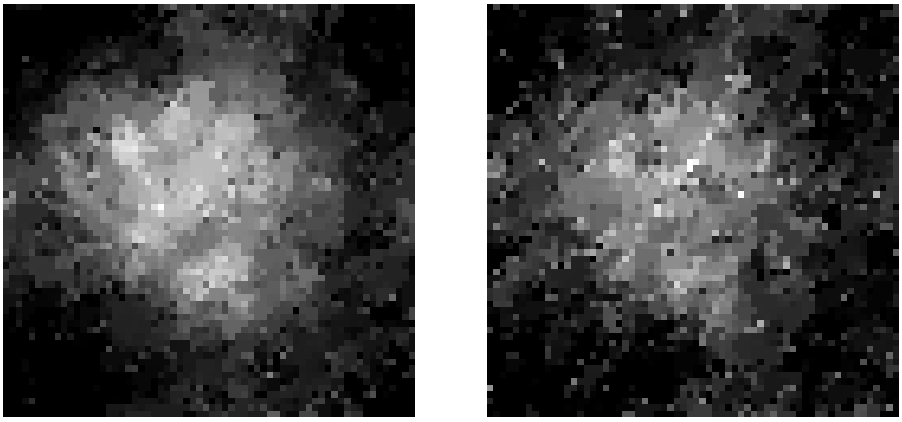


(a)



(b)

Figure 4.7: *Median (fig. 4.7a) and MAD (fig. 4.7b) of a reconstruction's pixel relative error plotted against the gaussian noise's VSR added to the measurements of the simulated observation.*



(a) *W=2048%,* `tvqc_logbarrier.m`. **(b)** *W=8192%,* `tvqc_logbarrier.m`.

Figure 4.8: *Reconstructions of fig. 4.1 for simulated observations where with: maximum 'photon' flux of 1000 per pixel; Poisson noise; 1500 measurements; and gaussian noise with W VSR.*

4.2 Implementation

Considering the results obtained in the previous section, the signal reconstruction subsystem was implemented as the `ReconstructImage.m` script (see appendix C.15) employing the `tvqc_logbarrier.m` algorithm; for convinience, the previous script is called by the `ReconstructImageFromFiles.m` wrapper (see appendix C.14).

The solvers used in `tvqc_logbarrier.m` are optimized to use the graphics processing unit (GPU), thus being prepared to parallelize the computational load, unlike most public C/C++ linear algebra libraries. Even though some of these libraries, like the `Eigen` library[61], are capable of multi-threading (distributing the load over multiple central processing unit (CPU) cores), but that process isn't as efficient as using the GPU.

Because the `ReconsctructImageFromFiles.m` script receives as inputs an observation vector and an explicit sampling matrix a program, `hadIndexToMatrix.cpp`, was written. This program expects to take as input a file of indices and from it, using the `Create_Hadamard_Lines()` function, writes a new file with the corresponding sampling matrix. The code can be consulted in appendix C.6.

Having successfully reconstructed images from simulated measurements, all that is left is for this subsystem to be tested with data from real measurements. Such proceedings are described in chapter 6.

Chapter 5

Development of a Structural Subsystem

In order to integrate and test the integration of the optical, signal coding and acquisition subsystems, in the kind of situation to which they were designed, it is required that this system can be coupled to a telescope.

A prototype mechanical support structure was therefore designed for COSAC. It was decided that it should weight less than 3 kg, should be able to withstand normal usage, coupled to an equatorial mount, while out in the open with temperatures ranging from 0 °C to 40 °C, and specular reflections within the instrument should be minimized in vicinity of the signal path.

The designed components include the structural frame, the casing, supports for components of the optical, acquisition and signal coding subsystems, baffles and a coupling flange. To do so, Solidworks was used.

5.1 Design

To meet the aforementioned requirements the frame was projected to use OpenBeams[63] and solid aluminum beams, and be coupled with bolts and screws to a 5 mm thick aluminum sheet. This sheet, the base, is a fixation support for most components. The casing and baffles were designed to use 0.5 mm aluminum sheets but, due to short supply, 1 mm sheets were used instead (the overall impact on the weight wasn't relevant). The coupling flange and Lightcrafter's support were produced out of solid aluminum, while the supports for the lenses and photodiode were 3d-printed out of photopolymeric resin.

5.1.1 Changes to the Optical Subsystem

The optical subsystem designed by Pires in her Masters' dissertation[21] is comprised of three lenses, the DMD and the Hamamatsu S1223 photodiode. Its configuration can be seen in fig. 5.1. All lenses are biconvex, 25.4 mm (an inch) wide and 3 mm thick at the edge, and while one of the lenses (L3) is 6 mm thick at the center, with a 50 mm focal distance, the other two (L1 and L2) are , with a 150 mm focal distance. The distances and angle marked in the picture are the following: d0 = 50 mm; d1 = 155 mm; d2 = 150 mm; d3 = 30 mm; d4 = 51 mm; $2\theta = 24°$.

This configuration was kept regarding the lenses used, the angle θ and distances d1, d2 and d4; d0 and d3 are irrelevant due to the signal rays being parallel in those regions and were modified when required.

A quick search for optomechanical supports on Thorlabs website rendered some options as

those presented in tab. 5.1. Given that options such as these didn't fall within our budget, and that a 3d-printer and resin was available in our laboratory, it was decided to explore a more economical approach.

Product	Code	Qtt.	EUC (€)	ETC (€)
Option 1				
∅25.4 mm Lens Mount with SM1 Internal Threads and No Retaining Lip, 8-32 Tap	SMR1	3	17.48	52.44
Adjustable Lens Mount: ∅7.1 mm to ∅45.7 mm, M4 Tap	LH1	1	39.80	39.80
Dovetail Optical Rail 75 mm	RLA075/M	2	27.22	54.44
Dovetail Rail Carrier, 15.2 mm×25.4 mm, #8 (M4) Counterbore	RC4	4	23.78	95.12
			Total	**241.80**
Option 2				
∅25.4 mm Optic Mount for Use with CT1 MS Stages	CT101	3	17.48	52.44
12.7 mm Dovetail Translation Stage, M4 Taps	DT12/M	4	74.26	297.04
Adjustable Lens Mount: ∅7.1 mm to ∅45.7 mm, M4 Tap	LH1	1	39.80	39.80
Top Plate for DT12 Stages, M3 and M4 Tapped	DT12CTA/M	3	32.00	96.00
			Total	**485.28**

Table 5.1: *Estimated cost of building an optomechanical support system for the lenses, using two distinct setups of Thorlabs parts.*

The lenses' supports, the support for the photodiode and the support for Lightcrafter though were redesigned and adapted to minimize the dimensions of the instrument as a whole.

It was decided that the structure to hold the supports for the lenses and photodiode should offer one degree of freedom, so that the instrument's focus can be tunned should there be a deviation of the lenses' nominal focal length. To accomplish this a set of linear rails (these rails, 80 mm and 140 mm long, were sawed from a 300 mm rail) and ball bearing carriages[62], was used (see fig. 5.2). The supports for the lenses and photodiode were then designed according to their dimensions as well as those of the carriages. Though some carriages with an included brake were also available for purchase they didn't fit the budget so, external brakes were also designed. These can also slide along the linear rails but include a cavity in which a nut can be fit and fixed, a bolt can then be attached and lock the brake into place. Placing a brake on each end of a carriage will then keep it from moving.

The cost of the resulting system can be consulted in tab. 5.2. The technical drawings for the designed and above described components can be found in appendix D.2.2.

The support for the Lightcrafter was designed taking its dimensions and the height of the rail and lenses' support assembly, as to have the center of the DMD at the same height as the center

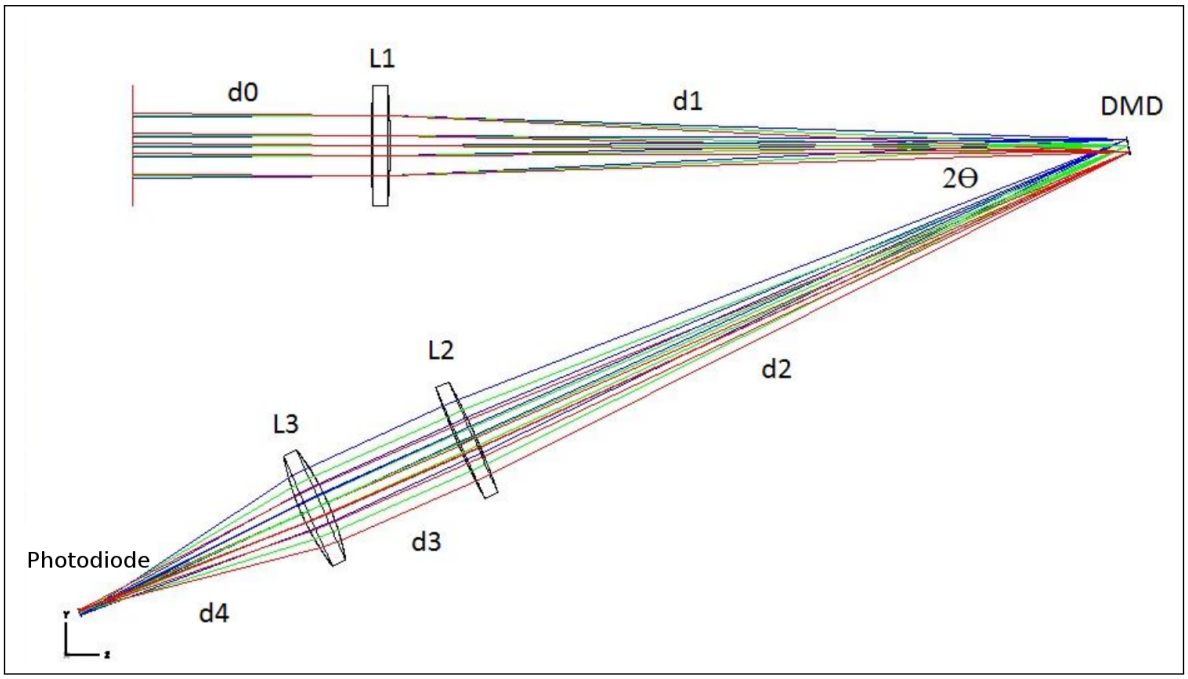


Figure 5.1: *Configuration of the optical subsystem designed by Pires (adapted from [21]).*

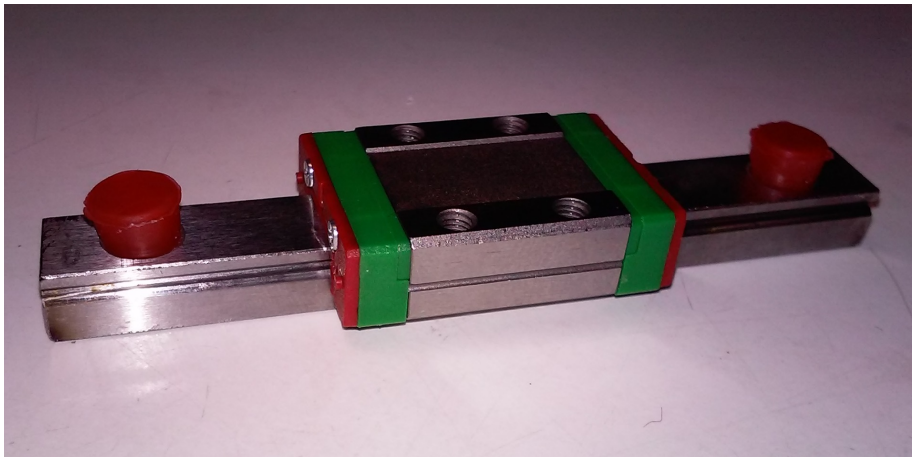


Figure 5.2: *Linear rail and ball bearing carriage by MakerBeam.*

of those lenses and photodiode. After accounting for the dimensions and fixtures, the bottom of this support was hollowed out (as can be seen in appendix D.2.1) to reduce its weight.

To reduce diffusion and specular reflections of the signal, within the casing, along its path and pollution of the signal by the leds that are embedded in the electronic components, three baffles were added to the design of the optical subsystem (see appendix D.2.1), one to block the light coming from the area that'll hold the acquisition and signal coding subsystem and two others parallel to the signal path until a bit before the point where the signal is reflected (and coded) by the DMD, this can be seen in fig. 5.3. To improve their efficiency they, along with every other aluminum component used in this project, underwent black anodizing.

5.1.2 Other Structural and Support Elements

To couple COSAC to a telescope a flange was designed with a 3 mm thick rectangular base and a 22 mm tall tube with an inner diameter of 25.4 mm and an outer diameter of 50 mm.

Product	Qtt.	EUC (€)	ETC (€)
Photopolymer Resin, black, 1 L	0.06	163.35	9.80
Linear slide carriage (MakerBeam)	2	13.25	26.50
Linear slide rail (300 mm) and carriage (MakerBeam)	1	21.50	21.50
		Total	**57.80**

Table 5.2: *Approximate cost of production of the optomechanical parts necessary to support the lenses and photodiode. The required resin amount was doubled to account for wasted material.*

The structure is comprised of: four 270 mm OpenBeams[63]; one 230 mm OpenBeam (attained from sawing a 270 mm OpenBeam); four 40 mm OpenBeams(attained from sawing four 45 mm OpenBeam); four corner cubes[64]; two 230 mm beams, designed to fit the remaining structure while providing support to the flange; a plate, to serve as base, which, aside from providing fixture spots to most of the structure and remaining subsystems, has engraved on its surface markings to align the carriages with in order set the optical subsystem with the distances mentioned in section 5.1.1; two holding pieces that were added to the structure to provide extra fixture spots to the back panel of the casing; two lateral panels; a front panel; a back panel; and a cover.

The base design was intended to be simple and such that it could accommodate the optical subsystem configuration and the acquisition and coding subsystems, it was decided to use a 5 mm thick aluminum plate for this component is to bear most of the mechanical load. The casing panels were designed in 0.5 mm thick aluminum sheet to keep the instrument light, even though it will be more vulnerable to mechanical vibrations.

Schematics for the designed components can be found in appendix D.2.1.

The resulting ensemble measures 320×260×60 mm, with an estimated weight of 2.5 kg (not accounting for pIDDO, bolts, nuts, lenses, cables and IEC plug). A render of the mechanical structure with elements of the optical, acquisition and signal coding subsystems can be seen in fig. 5.5.

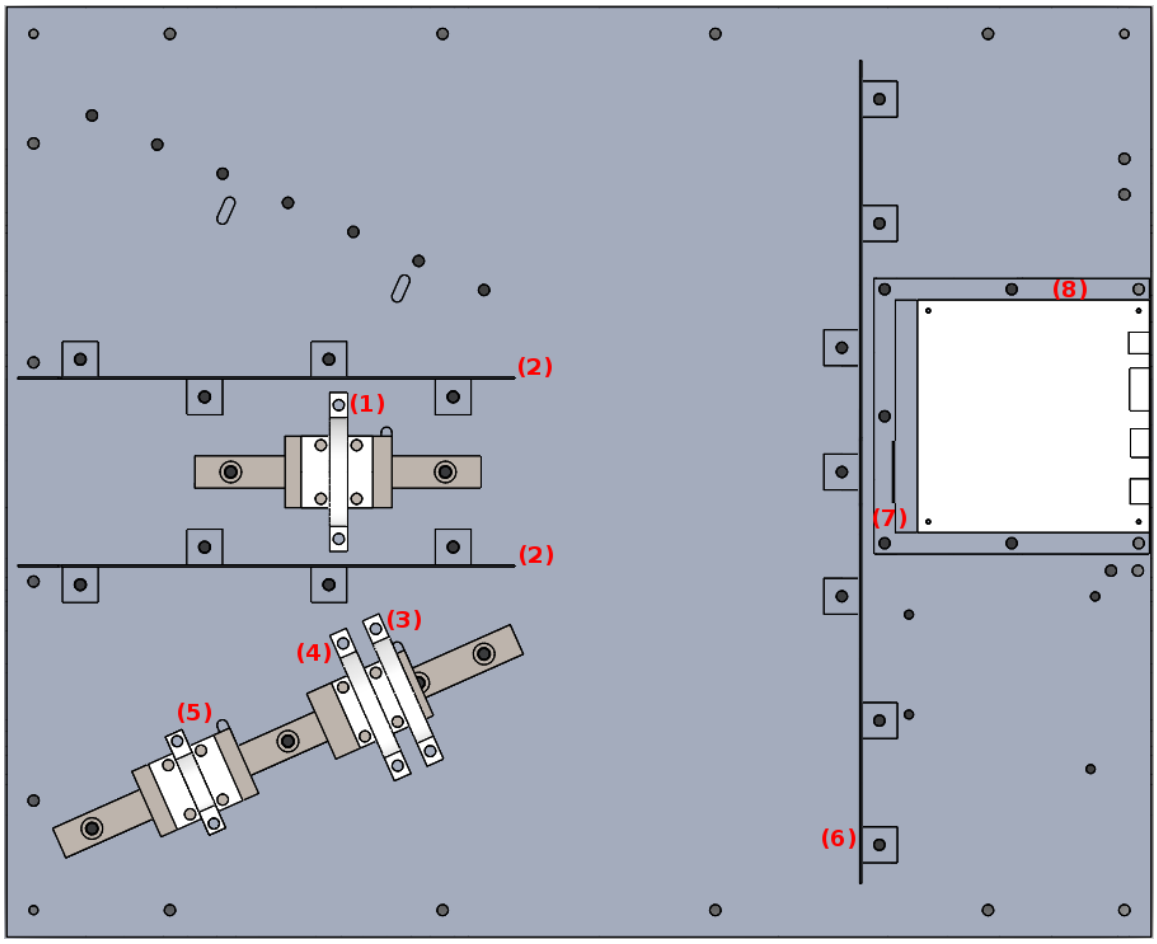


Figure 5.3: *Top view of the configuration of the optical subsystem applied to COSAC.* ***(1)*** *L1;* ***(2)*** *signal path baffles;* ***(3)*** *L2;* ***(4)*** *L4;* ***(5)*** *Hamamatsu S1223;* ***(6)*** *electronics' baffle;* ***(7)*** *DMD;* ***(8)*** *DLP Lightcrafter.*

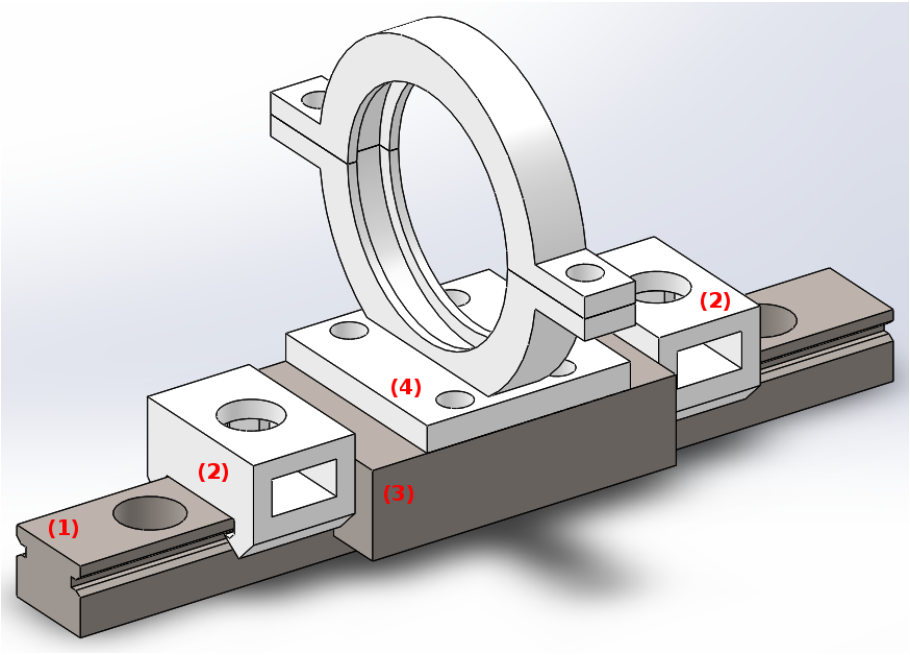


Figure 5.4: *Rendering of the support set for the lenses.* ***(1)*** *linear rail;* ***(2)*** *stoppers;* ***(3)*** *carriage;* ***(4)*** *lens' support.*

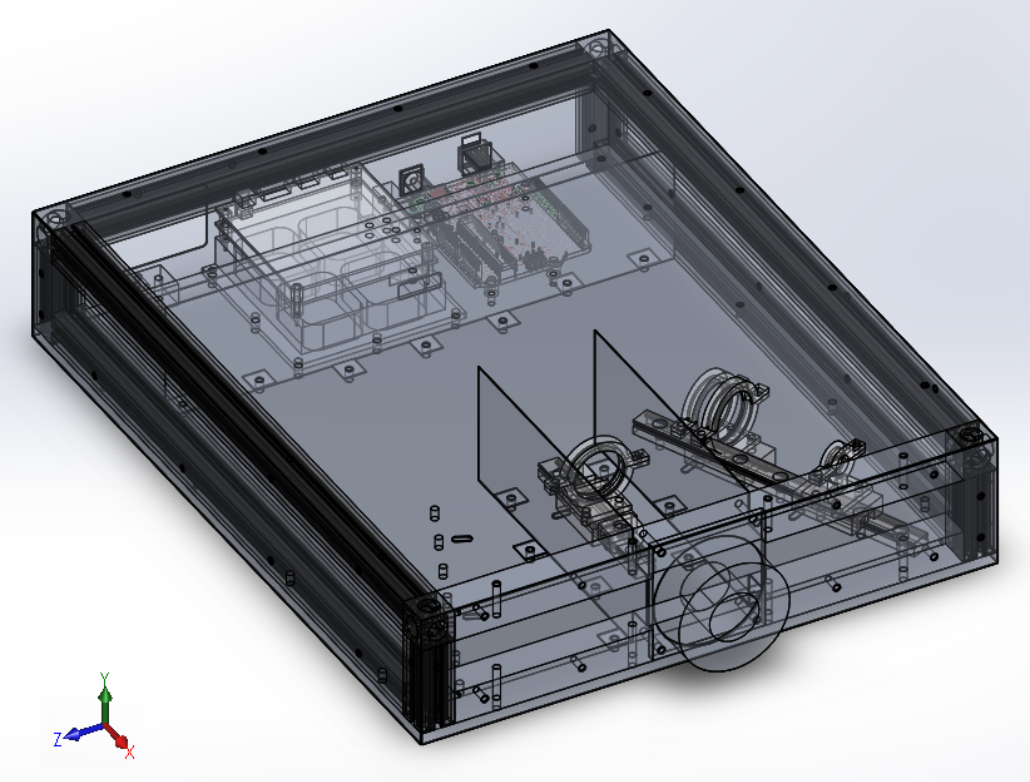


Figure 5.5: *Transparent rendering of COSAC's mechanical structure, inside renderings of elements of the optical, acquisition and signal coding subsystems can be seen.*

5.2 Finite Element Method

5.2.1 Introduction

When analytical methods (AM) to describe a system don't exist or are too slow providing an exact solution due to the complexity of said system, numerical methods (NM) are used instead to find an approximate solution within a fraction of the time it takes to use AM.

One of such NM is the finite element method (FEM), in which the complex problem is subdivided into smaller simpler problems, called finite elements, resulting in a system of algebraic equations that yield an approximate solution of the system at a discrete number of points across its domain. These partial solutions are then combined to describe the whole system. Methods based on the calculus of variations and which often employ as principles either the theorem of minimum potential energy or the virtual work in the context of materials behaving in a linear-elastic manner (when employing the principle of virtual work nonlinear behavior can also be described), are then used to approximate a solution by minimizing an associated error function. For structural problems the solutions usually concern the stress applied to each element and the displacement of each node (a point on the interface of elements) when a load is applied[65].

To validate the designed structure, prior to fabrication, a FEM analysis was done using Solidworks' simulation tools.

5.2.2 Static Simulations

Having designed the structure and assigned materials to every part, it was decided to run some simulations in order to determine whether or not this structure will endure the mechanical stresses it will be subject to (under different temperatures) and whether or not the deformations that'll occur have a low enough magnitude as to not impair the optical path. The temperature simulations were realized first and their results were applied to static simulations as thermal effects. These aside the only other external load on the model was gravity.

In all the realized simulations the a fixture was applied on the outside cylinder surface of the flange, emulating the telescope mount fixture system. The simulations that were run emulated the structure being held in 5 different angular positions, achieved by changing the direction of the gravity vector in the simulation ((δ,ϵ,ζ)={(90°,0°,90°), (45°,45°,90°), (0°,90°,90°), (90°,45°,45°), (45°,45°,45°)})[1], and, for each position, a simulation was run with 5 different temperatures (T=0,10,20,30,40 °C).

A solid curvature based mesh was used with elements of size ranging from 2.893 mm to 14.466 mm (see fig. D.2 in appendix D.5.1). To determine whether a finner mesh would yield better results control simulations were realized, using a mesh of the same kind but with elements ranging from 0.589 mm to 2.893 mm (see fig. D.3 in appendix D.5.1), for the following cases: T=0 °C, (δ,ϵ,ζ)=(90°,0°,90°); T=0 °C, (δ,ϵ,ζ)=(45°,45°,90°); T=20 °C, (δ,ϵ,ζ)=(0°,90°,90°). The difference in the yielded results ranged from 0.5 % to 1% and was therefore deemed unworthy to pursue since the processing time was 600% larger.

After all simulations were done, with the first mentioned mesh, the displacement levels in the optical region were evaluated and the global maximum displacement value for each run was registered. These values were then plotted against the temperature (making 5 sets of fixed position). The results, which plots can be seen in fig. 5.6, show that the maximum displacement

[1] The set (δ,ϵ,ζ) denotes the angles that a vector makes with, respectively, the X¸Y and Z axes.

barely surpasses 0.25 mm in normal working conditions, since the acquisition system measures the signal power over a time period instead of the signal spacial distribution, this doesn't introduce large distortions on the reconstructed image, as such the structure was approved for fabrication.

Renderings of the simulations at T=0, 10, 20, 30, 40 °C can be found in section D.5.

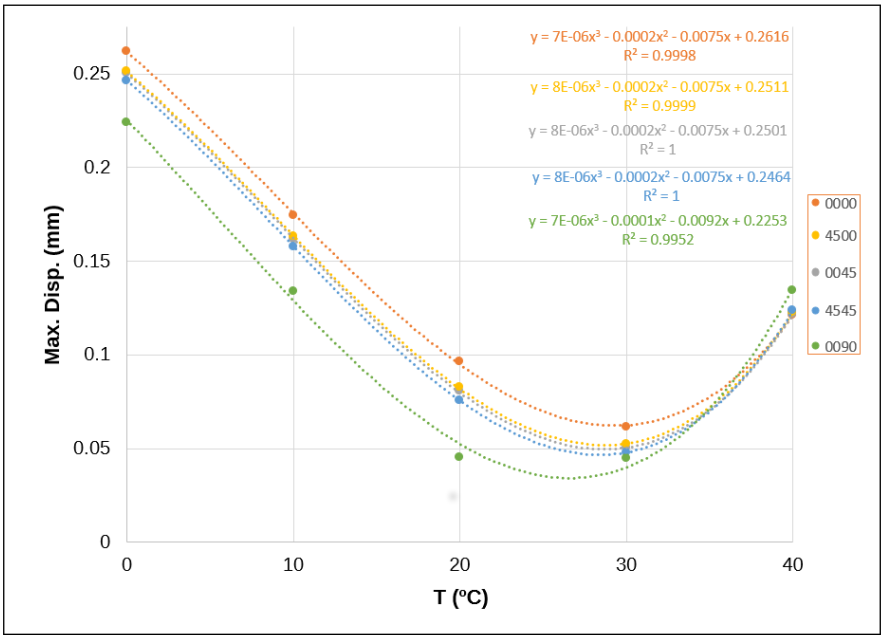


Figure 5.6: *Plots of temperature (X-axis) vs maximum displacement (Y-axis) taken from simulations using 5 different directions for the gravity vector.*

5.3 Fabrication

The supports for the lenses and photodiode, as well as the stoppers, were produced in the laboratory using black photopolymeric resin by a stereolithographic 3d printer, a Form 1+. They were then cleansed using isopropyl alcohol and submitted to a cure inside an UV light modified oven. Afterward they were clipped from their supports and sanded to remove the clipping joints and excesses (resin that didn't solidify properly upon printing and flowed to other regions). These pieces can be seen in fig. 5.7. Due to a lack of resources, some of the stoppers were printed in transparent resin and painted black after curing.

All the designed aluminum components were machined in the department's workshops using a 3-axes CNC, a 3-axes milling press and a drilling press, among other regular workshop tools. Due to lack of resources it was not possible to fabricate the baffles and panels with 0.5mm aluminum sheet, 1mm sheet was used instead. After milling, drilling, threading and sanding was over, all aluminum components were dipped into isopropyl alcohol in an ultrasound machine to wash off the dust. These pieces can be seen in fig. 5.8. They were then sent to be black anodized at Alfa Sul Alumínios do Sul, S.A..

After the aluminum pieces were anodized the casing was assembled to check its junctions and overall structure (see fig. 5.9). While checking it came to notice that the hole for the flange on the frontal case sheet was too wide ∼2 mm too wide, because of that, to avoid light leaking in through it, black tape was used to cover the gap.

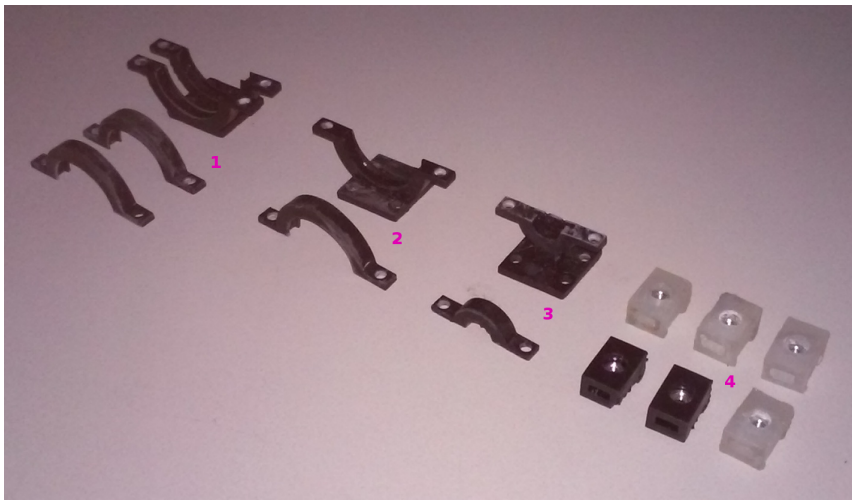


Figure 5.7: *The designed resin components after printing, curing and sanding.* ***(1)*** *Bottom part of support for two lenses and respective top caps;* ***(2)*** *bottom part of support for one lens and top cap;* ***(3)*** *bottom part of support for Hamamatsu S1223 and top cap;* ***(4)*** *stoppers.*

Having checked the box, a telescope was assembled to the box to check the coupling (see fig. 5.10), and it was a success.

With the structural subsystem ready, COSAC could then be assembled. In the next chapter we'll describe the procedures involved to integrate all subsystems, validate the instrument and characterize it.

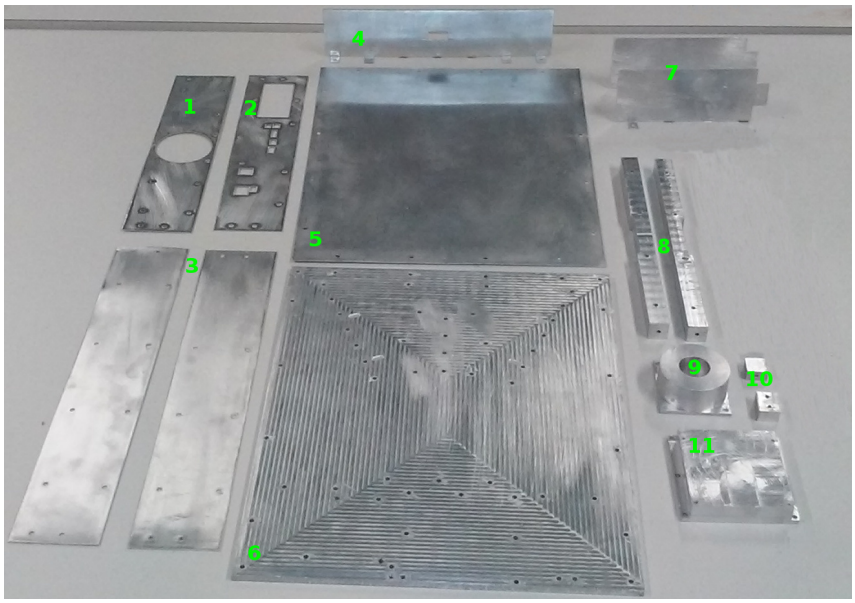


Figure 5.8: *The designed aluminum components after carving and sanding.* ***(1)*** *Front panel;* ***(2)*** *back panel ;* ***(3)*** *side panels;* ***(4)*** *baffle for electronics area;* ***(5)*** *cover;* ***(6)*** *base;* ***(7)*** *baffles for light signal input area;* ***(8)*** *front beams;* ***(9)*** *flange;* ***(10)*** *holders;* ***(11)*** *support for DLP Lightcrafter.*

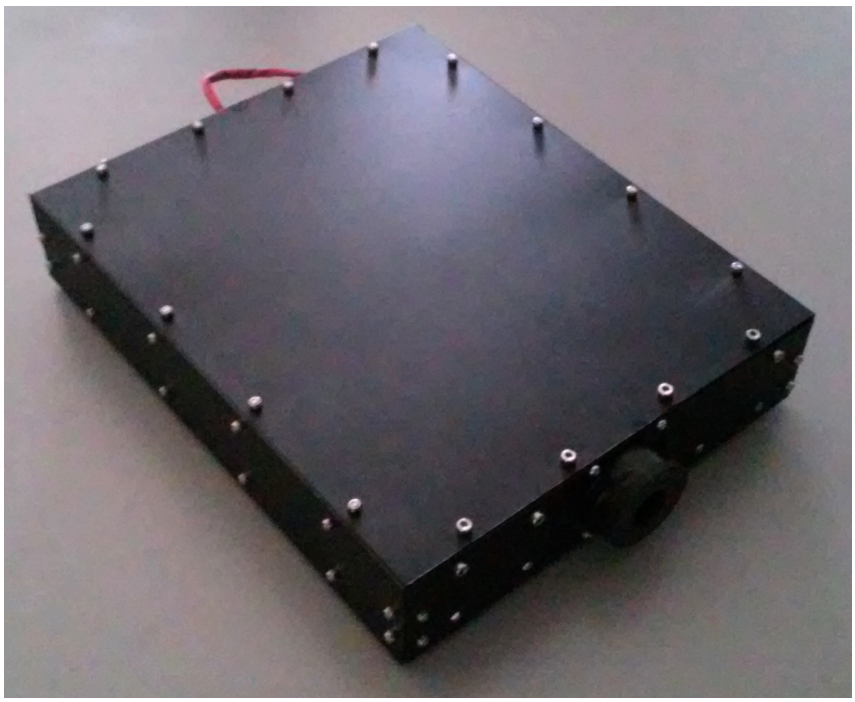


Figure 5.9: *COSAC's casing assembled.*

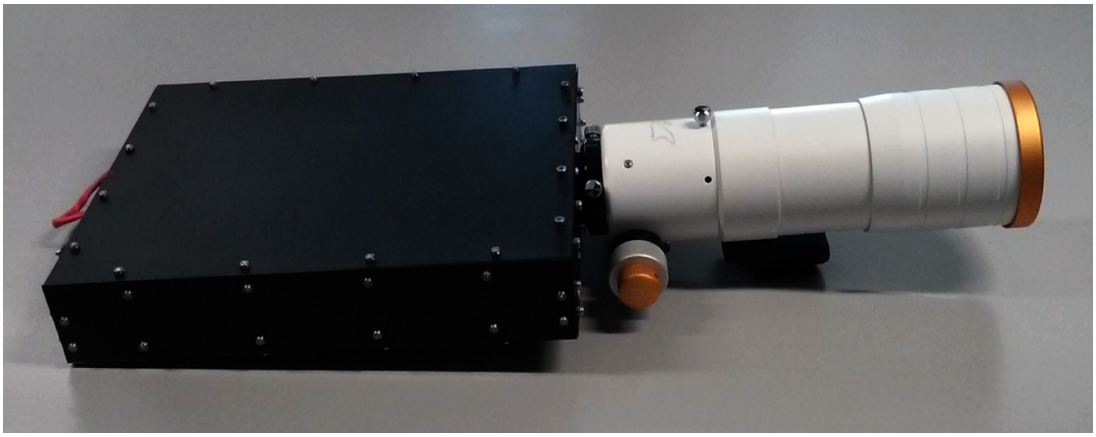


Figure 5.10: *A small telescope coupled to COSAC's casing.*

Chapter 6

Integration, Validation & Testing

In the previous chapters we've described, more or less extensively, the architecture and development of every subsystem that integrates COSAC (for more details: on the optical subsystem, see [21]; on the acquisition subsystem, see [20]). We shall now detail the proceedings pertaining the integration of each subsystem and COSAC's calibration and analysis. At the end of the chapter, comparative results between three acquisition modes will be presented.

6.1 Integration

Having all subsystems working, individually, as envisioned, it was then necessary to connect and adapt them to each other so as to achieve our goal of a functioning COSAC.

6.1.1 Establishing Communication between the Acquisition & Modulation Subsystems

So that a measurement is made for each pattern displayed it is necessary that the coding and acquisition subsystems communicate to secure their synchronization. This communication is mediated by the Arduino, which talks to the control program `DMD-CS.cpp`, running on the PC, via Serial USB and sends triggers to the DLP's FPGA via TTL. In its turn, as already described in chapter 3, the `DMD-CS.cpp` directly communicates with the DLP's DM365 via USB over net, in order to properly control the DMD's configuration.

In order to establish TTL communication that the raster signal socket was added to the pIDDO pcb. This socket was connected to the contact pins that plug into the Arduino according to the description regarding the trigger connector[43] (see fig. 2.5). Then a 4 wire cable, held by a thermoretractile sleeve and double ended by raster signal plugs, was built (see fig. 6.1).

With the hardware ready, some modifications to the software had to be made. Since the Arduino will be giving triggering (by communicating with the FPGA) the pattern switch it is then required that it knows when to start sending those triggers, how many triggers it has to send and how soon it can send a trigger after the last one. To achieve this, some instructions using functions of the `rs232.h` library were placed in `DMD-CS.cpp`, which has already been described in section 3.3.3, and some edits were done to the code running on the Arduino (which original version can be found in [20]), the resulting program was named `pIDDO.ino`.

Figure 6.1: *Raster signal cable that was assembled so that the Arduino could send TTL triggers to the DLP's FPGA.*

Changes to the Acquisition Control Program

The acquisition control program, `pIDDO-24bit.ino`, is, as mentioned previously, based on the code presented by Bandarra in her dissertion[20] for the same purpose. This in turn uses code made available by Beale to properly control an LTC (`pIDDO-24bit.ino`) using an Arduino[39] so that together with the instructions to control the IVC and the SD card a proper measurement can be acquired.

Some changes and additions to those sets of instructions were made to not only achieve the aforementioned synchronism between the acquisition and control subsystems but also to reflect the changes made to the acquisition circuit, to reduce the runtime of each loop and allow several acquisition series to be made without losing the results of previous series.

Some of the variables concerning Arduino pin assignment were changed (`S1`, `S2`, `CS_ADC`, `CS_SD`), others were created (`BUSY_ADC`, `CD_SD`, `TRIG_OUT`, `TRIG_IN`, `volts_A`, `volts_B`), and `resetPin`[20] was removed. Other created variables include `String`s (`prefix`, `suffix`, `fileName`) used to build the filename[1] in which the acquisition series will be saved, and an `int trigNum` which will hold the number of triggers, sent by `DMD-CS.cpp` via Serial USB, to send to the DLP.

In the `setup()`, after initializing the Serial port, using `pinMode` to define which pins will deal with inputs or outputs, setting both chip selects to `HIGH` and `TRIG_OUT` to `LOW`, the program waits for input from `DMD-CS.cpp` to continue. Then the SD card connection is initialized and a loop is run to verify the existence of previous measurement series files and then create a new file with a different name. To conclude the setup `S1` and `S2` are set to `HIGH` (`S1` and `S2` control the IVC's switches S1 and S2, respectively, which are used to control the integration process[33]).

Entering the `loop()` the program waits for `DMD-CS.cpp` to request the name of the file where the measurements will be stored. After sending that information, `pIDDO-24bit.ino` then waits for the integration time of each measurement and the total number of measurements, to be made in the present run, to be sent by `DMD-CS.cpp`; if these numbers are in the expected format[2] they are parsed into `intTime` and `patNum`respectively, if not the loop restarts. In every case a message is sent to `DMD-CS.cpp` to report on the outcome of the previous transaction.

A `while()` loop then starts to run until a counter reaches the value of `patNum`. In each iteration the program waits for `DMD-CS.cpp` to input the number of triggers that will have to be

[1] This filename shall have the following structure: `prefix`+i+`suffix`, i $\in \mathbb{N}$

[2] Information traded using the Serial port is split into bytes, as such most ready-to-use Serial libraries are prepared to send and receive streams of `char`. In the present application, to ensure that only the intended information is parsed by both `pIDDO-24bit.ino` and `DMD-CS.cpp`, messages with length greater than 1 byte have a header, '?', and a tail, '!', to signal both their beginning and ending.

sent to the DMD, with that number being parsed into `trigNum`, if it is in the expected format, and a message is then sent to `DMD-CS.cpp` regarding the outcome. The counter is incremented the value of `trigNum` and a `for()` loop where the measurements will be realized is initialized.

This loop is iterated `trigNum` times and the following actions, which are based on the manufacturer's recommendations for switched-input measurement technique[33], take place:

1. A trigger is sent to the DLP for new pattern to be displayed;
2. IVC's switches are controlled to reset it and prepare for integration;
3. In an `nsample` iteration loop, the following instructions take place in order to make an average reading of the offset error:
   (a) LTC's BUSY pin is read and the program is held until its level is `LOW`;
   (b) The LTC is prompted to make a reading - which will be regarded as an offset error - using the `SpiRead()` function[39] (the code for this function can be consulted in appendix C.12);
4. IVC's switches are controlled once again to start integrating;
5. After a hold time defined by the user through `DMD-CS.cpp`, the IVC's switches are set to stop the integration and prepare for reset;
6. In another 10 iteration loop, the following take place in order to make an average reading of the signal:
   (a) LTC's BUSY pin is read and the program is held until its level is `LOW`;
   (b) The LTC is prompted to make a reading using the `SpiRead()` function[39] (the code for this function can be consulted in appendix C.9);
7. The difference between the signal and offset is then written to the file created on the SD card.

Once out of the previous loop, if all measurements were successfully performed, a one byte message is sent to `DMD-CS.cpp` for it to send the next value to be assigned to `trigNum`, and the process repeats until the `while()` condition is falsified thus concluding the acquisition series.

The new data filename is created and `pIDDO.ino` returns to wait for a new value for `patNum` to restart the whole process.

The resulting code, present in `pIDDO-24bit.ino`, can be consulted in appendix C.9.

The Arduino control program for pIDDO-10bit, `pIDDO-10bit.ino`, runs similarly to `pIDDO-24bit.ino`, differing only on the instructions used to read the IVC's integration result. The code can be consulted in appendix C.8.

Additionally, a flowchart of both `pIDDO-10/24bit.ino` can be consulted in appendix C.7.

6.1.2 Assembling COSAC

With the mechanical structures described in chapter 5 a casing for COSAC was assembled using bolts and self-locking nuts, with the parts for the optical and acquisition subsystems being mounted on the plate or respective support. As previously mentioned, the leds and lenses of the DLP were removed as those components were superfluous in this project.

To connect the electronic components to each other, sleeved cables were constructed using 0.75 mm wires and thermoretractile sleeves.

Since the power source used to power pIDDO had no casing, a plastic box was bought and holes in it were drilled to fix both its circuit board and solenoid, and also for the cabling to enter and exit it (see fig. 6.2). Furthermore the power source output cable was then ended with an IEC plug.

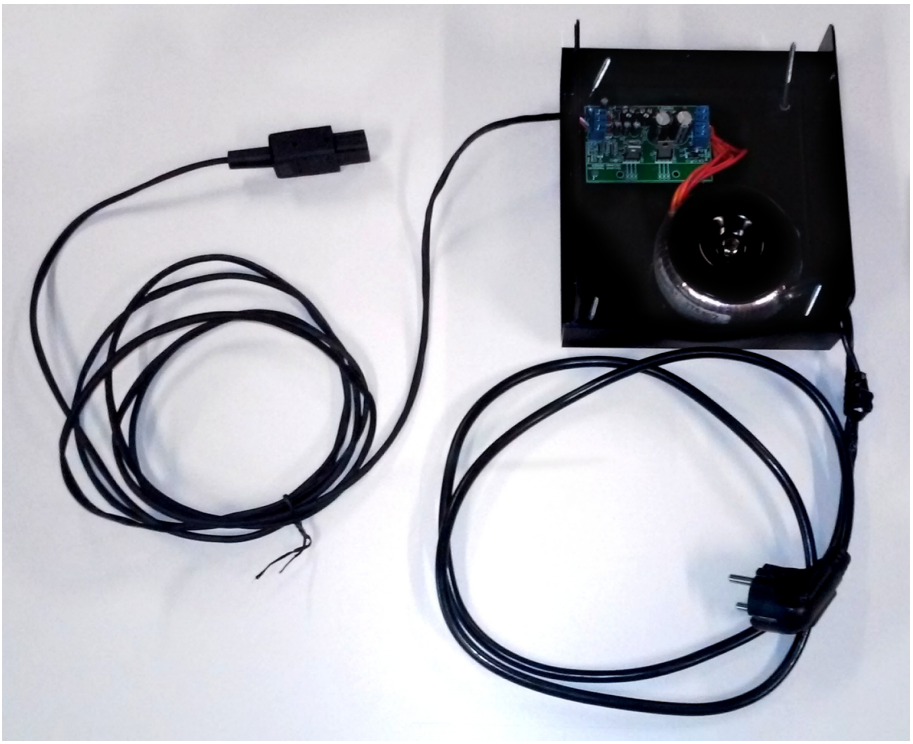


Figure 6.2: *Power source and cabling mounted in open casing.*

An IEC socket (with two fuses) was mounted on the back panel of the casing (see fig. 6.3). Its terminals were then connected to the pIDDO board using one of the previously described cables.

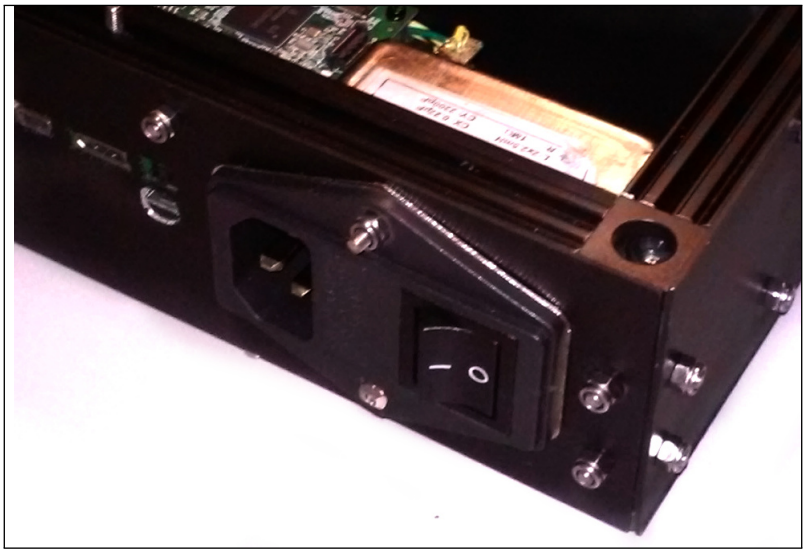


Figure 6.3: *IEC socket mounted in the back panel of COSAC's case.*

With all parts in their places the remaining mechanical parts could be mounted to form structure of the instrument (see fig. 6.4), then all that remained was for the front panel, side

panels and lid to be bolted to close the casing. COSAC was then weighted to be 2,883±0,001 kg, which is bellow the maximum weight defined in chapter 5.

In appendix E an estimate of the cost of production of all COSAC's subsystems is presented.

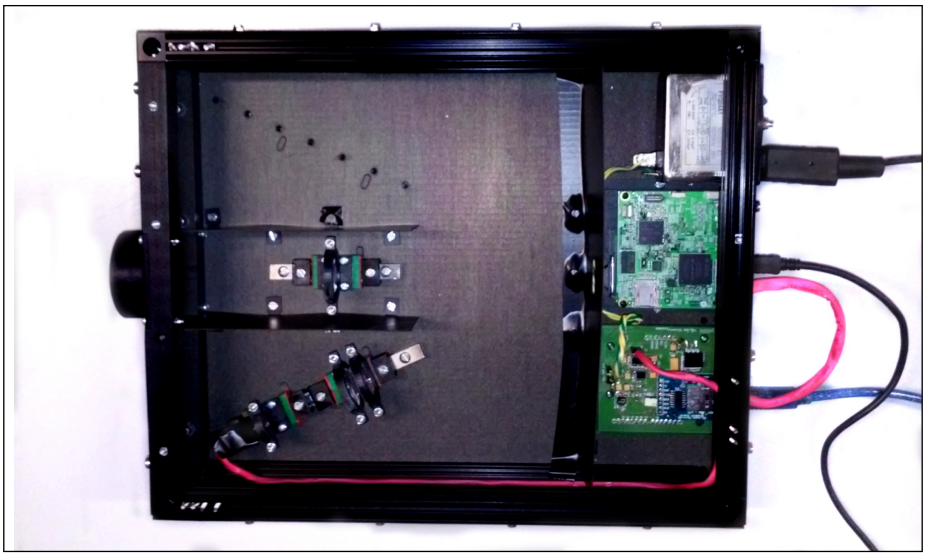


Figure 6.4: *Top view of the inside of COSAC with the optical and acquisition subsystems mounted on the mechanical structure.*

Aligning the Optical Subsystem

Even though every structural component was planed to ensure that the optical subsystem was assembled according to the design by Pires[21], upon mounting the LightCrafter in its support it came to notice that not only were holes on the two boards, present for bolts to pass through, wider than those of the support (making its fixation less precise) but also that the attachment between the boards wasn't rigid, making that the exertion of more pressure on the top board (by tightening the bolts) leads to deformation of the bottom board, thus dislocating it. It was then necessary to not only confirm that the lenses were properly setup so that an entering light signal is properly focused on the photodiode after being reflected, but also to determine the optimal position of the DMD so that the entering signal is focused and centered on its surface and is reflected in the right direction.

The following equipment was used in this procedure:

- Blu-tack adhesive
- Kinematic Mount, KC1-T, Thorlabs
- Kinematic Mount, KM200, Thorlabs
- Laser pointer adapted to a circuit controlled by an Arduino Micro (developed in this lab)
- Lens tube, Newport
- Lens tube adapter, 12 mm to SM1, Thorlabs
- Lens tube adapter, 2 inch to SM1, Thorlabs

- Mirror, 20D20ER.1, Newport
- Optical breadboard, Edmund Optics
- Optical subsystem
- Optical table, Melles Griot
- PC running the LightCrafter's GUI
- Power source, TENMA 72-2535[52]
- Retaining rings, 1 inch, Thorlabs
- Structural subsystem
- Targets, 1 inch, Thorlabs

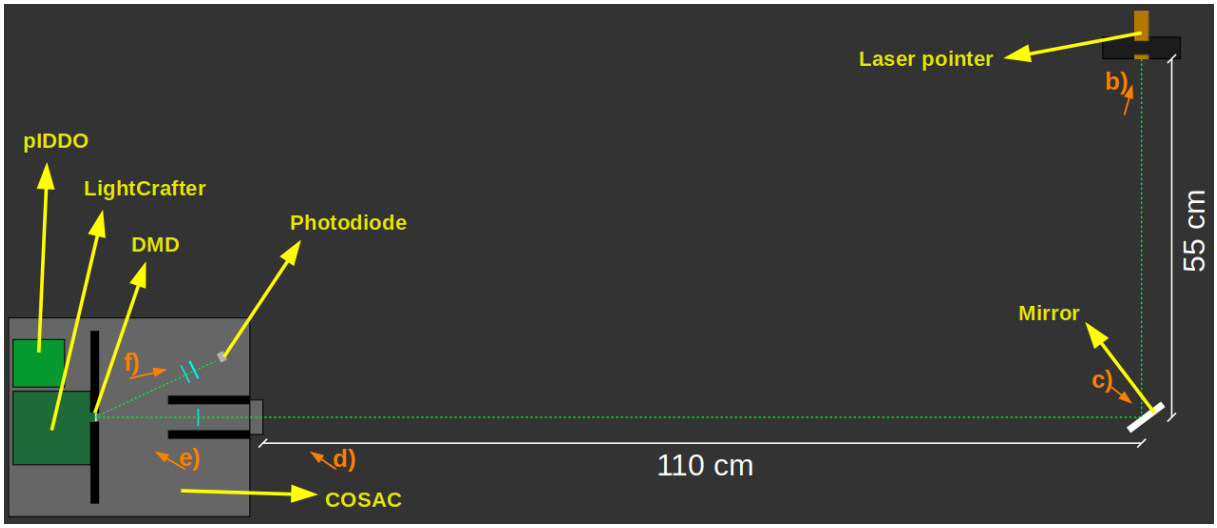
The optical subsystem was assembled within the structural subsystem, the latter was placed on top of an optical breadboard and positioned with the help of M4 Allen bolts; the breadboard was positioned on top of the optical table in the same manner. The mirror was mounted on the KM200 which was placed 110 cm across the front plane of the coupling flange, with the face of the mirror at a 45°(see fig. 6.5). The laser pointer was mounted in a 12 mm to SM1 tube adapter, which in turn was held by a KC1-T; this was placed 55 cm away from the mirror at a 90°angle with the flange-mirror direction. The distances used in this alignment were the greatest that were achievable using this setup; were it possible, to extend these distances would have increased the sensibility of the setup to angular shifts, thus providing more control over the distances during the alignment of the laser with the optical components up until it reaches the surface of the DMD. The laser pointer was controlled by an Arduino Micro and powered by the TENMA. The LightCrafter was connected to a PC running its GUI, with which it was configured to the "Test Pattern" display mode, with the DMD being set with the "Solid White" internal test pattern (which is equivalent to all micromirrors being tilted +12°).

A pinhole target was mounted on a 2 inch to SM1 tube adapter, which was in its turn mounted (using tape) to the flange. Other pinhole targets were placed in lenses' 1,2 and 3 steads. With the described setup, both kinematic mounts were used manipulate the direction of the laser beam to go through both pinhole targets, with the KC1-T being used as a translation module and the KM200 to adjust the tilt of the beam.

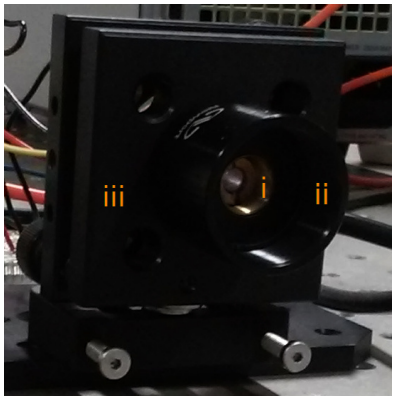
Using blu-tack, the position of the LightCrafter was shifted and tilted until the laser beam was centered on the DMD's surface and its reflection off of it would go through the remaining pinhole targets. Once these conditions were met the LightCrafter was locked in position with M2 bolts, with nuts and washers filling the necessary gaps.

Finally, to center the reflected beam in the surface of the S1223, the base of its bottom support was sanded and its fixation holes were elongated. Washers were again used to correct the height of the sensor, with its horizontal position being locked by the pressure of the fixation bolts.

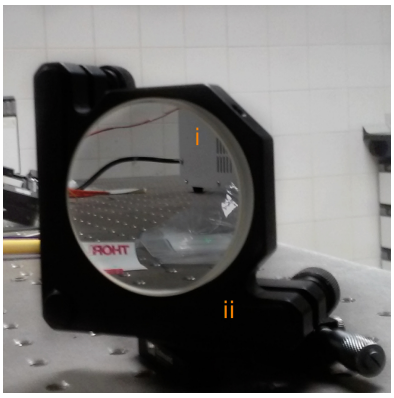
Afterwards, the targets were replaced by the lenses to fine tune the centering of the beam, now focused, on the surface of the photodiode.



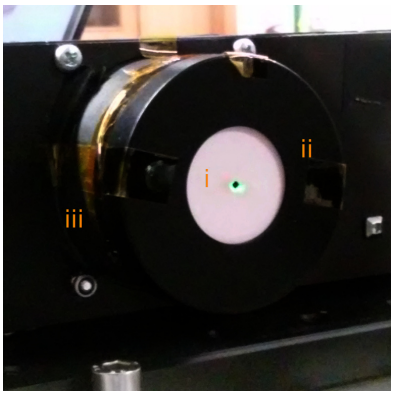
(a) *Schematic drawing of the setup used to align the optical subsystem.*



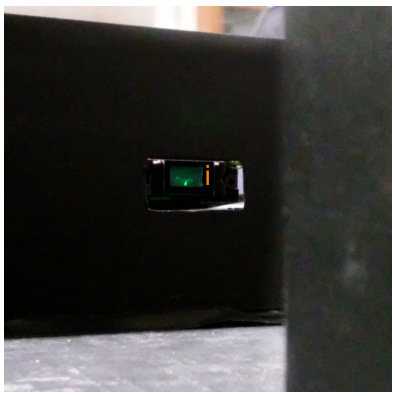
(b) *Laser pointer (i) mounted to a 12 mm SM1 tube adapter (ii) held by a KC1-T kinematic mount (iii).*



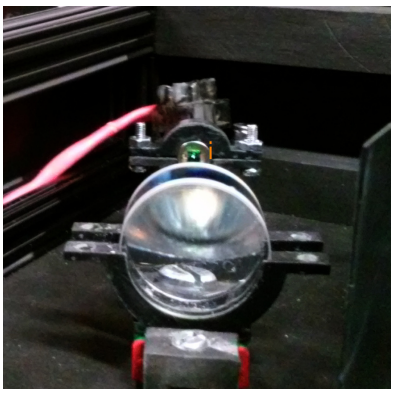
(c) *Mirror (i) held by a KM200 kinematic mount (ii).*



(d) *Pinhole target (i) mounted to a 2 inch SM1 tube adapter (ii), held by tape against the flange (iii).*



(e) *Laser beam being reflected by the DMD (i).*



(f) *Laser beam hitting the photodiode (i) after being reflected by the DMD.*

Figure 6.5: *Schematic and detail images of the setup used to align the optical subsystem. In fig. 6.5a the relative point of view for the remaining figures is indicated.*

6.2 Calibration

Having tested all the constituent elements that interact with the PC and the Arduino individually, and in pairs, with confirmation of their conformity to our objectives, we then tested the optical, coding and acquisition subsystems together.

For a first test of using the DMD and the S1223 photodiode together to encode and detect a light signal, it was decided to raster a target (by tilting a single micromirror, or a squared set of those, depending on the intended resolution), at a time, and evaluate the measured values. Since the patterns to be used to configure the DMD only use the DMD's central 256×256 square, it was in this same area that the raster was performed.

With that in mind, a new function was added to the `BMP.h` library, `Create_Raster_Lines()`. This function, receives as input an index `i` (which will correspond to i-th element of a square matrix), a pointer to an array which will hold the values of said matrix and that matrix's `rank`. The function assigns '0' to all of the array's elements except for the i-th element which is set to '1'. For this test `DMD-CS.cpp` was modified, having the instruction to generate a line of an Hadamard matrix (`Create_Had_Lines()`), replaced by the previously described function. This modified program shall be referred to as `DMD-Raster.cpp`.

To establish a baseline reading the same setup as the one described in section 6.1.2 was used, with the addition of the acquisition subsystem, where the Arduino ran a modified version of `pIDDO.ino` which, instead of writing to a file on the SD card, printed the measurements to the serial (which was observed using Arduino's serial monitor). With the "Solid White" pattern loaded on the DMD, the laser was turned-on and directed to S1223 to attain a normalized reading of its maximal output. Since the normalization interval's upper limit wasn't reached, pIDDO's potentiometer was tweaked to maximize the circuit's output (this was done for both 10 bit and 24 bit versions of the circuit). This procedure was repeated using a led lantern in the laser's place with the same outcome.

In this raster test, it was decided to use pIDDO-24bit since it is expected to yield better digitally resolved results. Four targets were used, a laser pointer, a laser cross line, a puncture pattern on black kapaline and a 'C' cut out also on black kapaline, both these two were lit up from behind by a white led lantern. Additionally, it was decided to use a 32×32 raster grid so as to be able to perform all raster tests in as similar conditions as possible (the cross line laser and flashlight are battery powered) without compromising too much resolution. The IT was initially defined as 100 μs, but was increased when required, and the sampling rate of the LTC was initially set at 880 Hz.

6.2.1 Laser pointer and cross line

While using the laser pointer as target, with the previous alignment setup, the obtained results were clear, with a spot showing on the image constructed, using the R environment, from the normalized measurements (see fig. 6.7a). The test was repeated for sampling rates of 1.76 kHz and 3.52 kHz and, upon comparing the results, it was decided to conduct the remaining tests at 1.76 kHz.

The laser cross line was provided by a level with a magnetic base. Two L-shaped steel supports were used to mount the level such that the laser cross line would be aligned with COSAC's aperture and the center of the DMD (see fig. 6.6).

The cross line initially yielded unexpected results, with one line, henceforth referred by as

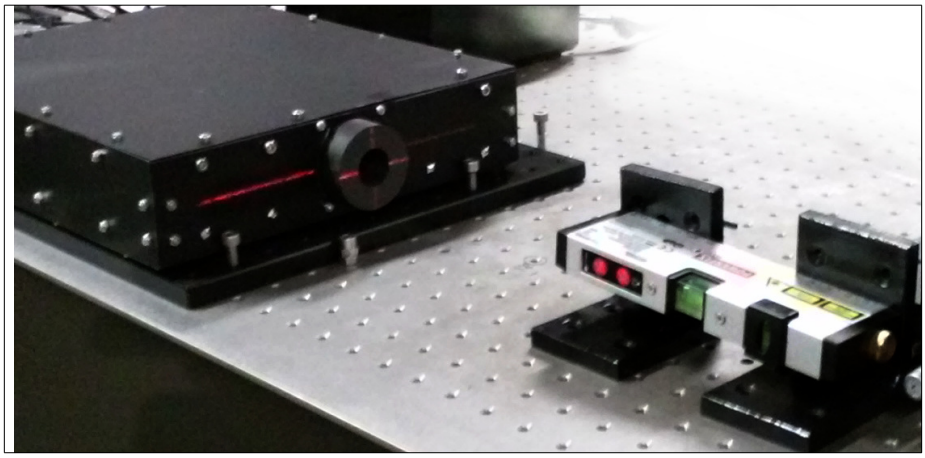
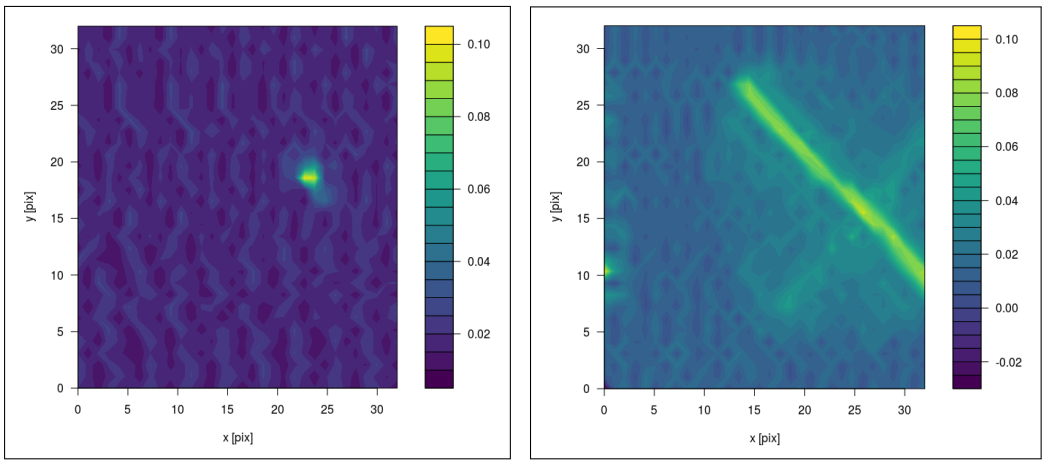


Figure 6.6: *Setup used to perform the raster of a laser cross line.*



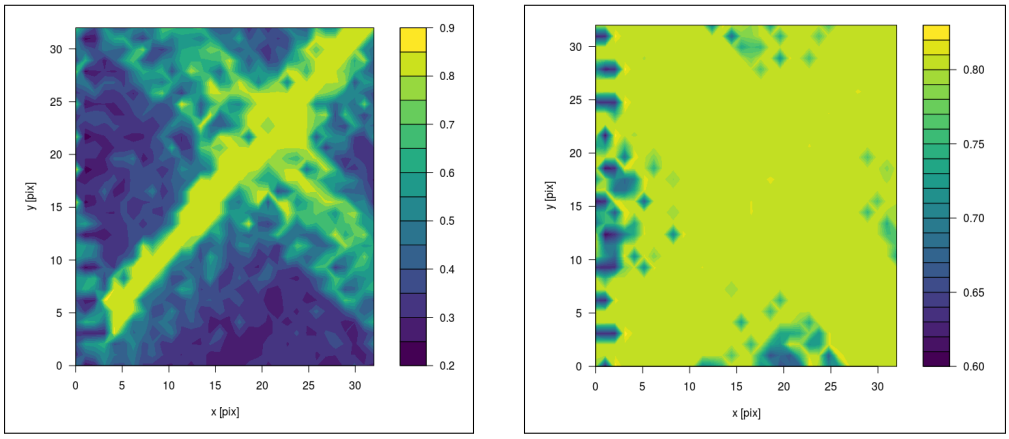
(a) **(b)**

Figure 6.7: *Filled contour of the reconstructed images from the resulting raster data (32×32 grid) of a laser pointer (fig. 6.7a) and a laser cross line (fig. 6.7b), using COSAC's optical and acquisition subsystems and* `DMD-Raster.cpp`*, with an integration time of 100 µs.*

the vertical, showing up as clearly as the previous trial's spot while the other line's presence was merely hinted (see fig. 6.7b). Because a difference in intensity (and spread) between the two was clear to naked eye it was decided to increase the IT to 400 µs and repeat the procedure. The new results, despite showing an increased contrast between the background and the vertical line, still didn't show the horizontal one. The laser cross line was then carefully aligned with the diagonals of the sampling area of the DMD, another raster was performed and still no horizontal line showed.

The IT was iteratively increased until the horizontal arm was discernible. This was finally achieved for 25 to 35 ms, below those values that arm wasn't visible and above the images start to become homogeneous across the whole grid (see fig. 6.8).

Following the initial results of the cross line raster, a raster with a 64×64 grid was performed in order to see if more details of the lines could be detected - the lines are actually diffraction patterns, small aligned segments that at a distance appear as a single line - and to determine whether the line thickness was due to the low resolution or poor focus. In the results (see fig. 6.9) the segmented nature of the line is clear even if distorted, the thickness however appears to be the same as in previous 32×32 rasters, which points to a poor focus.



(a) $IT = 30\,ms$ **(b)** $IT = 100\,ms$

Figure 6.8: *Filled contour of the reconstructed images from the resulting raster data (32×32 grid) of a laser cross line, using COSAC's optical and acquisition subsystems and* `DMD-Raster.cpp`*, with distinct integration times.*

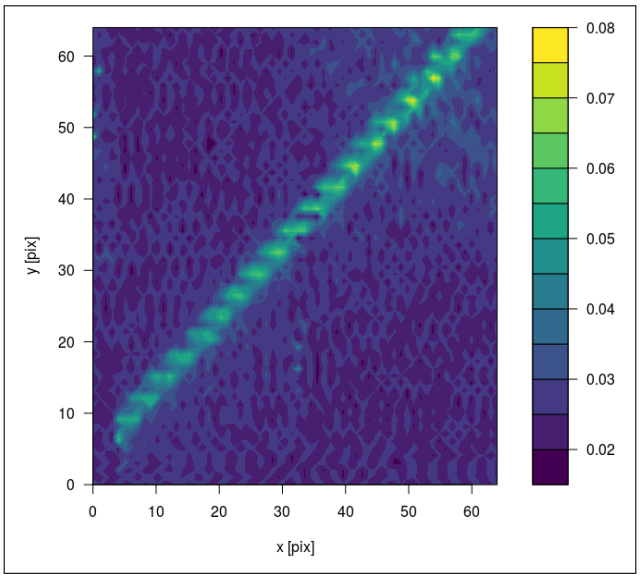
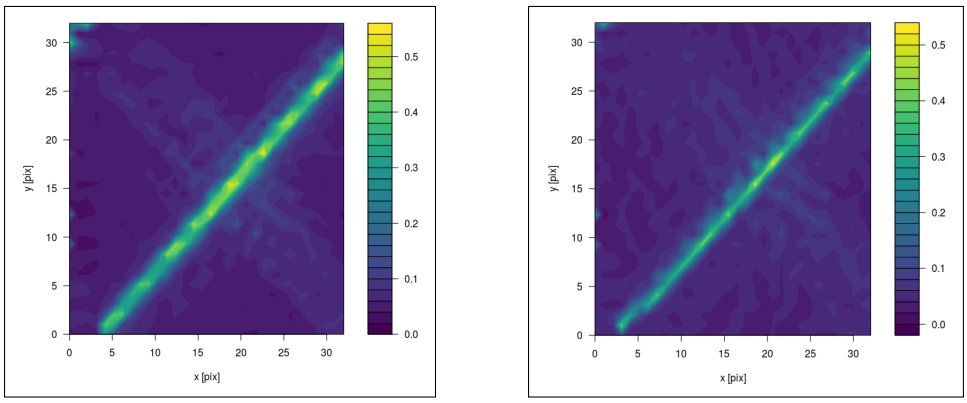


Figure 6.9: *Filled contour of the reconstructed images from the resulting raster data (64×64 grid) of a laser cross line, using COSAC's optical and acquisition subsystems and* `DMD-Raster.cpp`*, with an integration time of 100 µs.*

Focus Adjustment

In order to place L1 at the right distance to focus the entering signal on the DMD, 4 series of rasters were performed while varying the distance between L1 and the DMD. To measure the position of L1 relative to its original position a 150 mm ruler was set along the length of the rail on which L1's skate is supported.

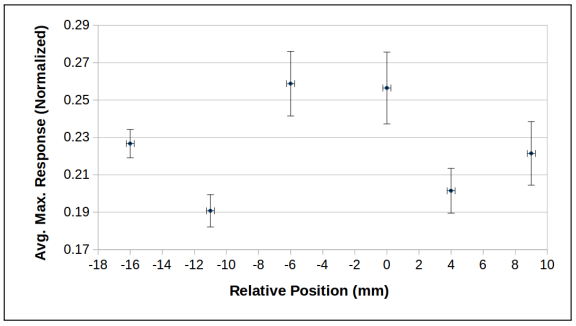


(a) $RP = 0\,mm$ **(b)** $RP = -5.5\,mm$

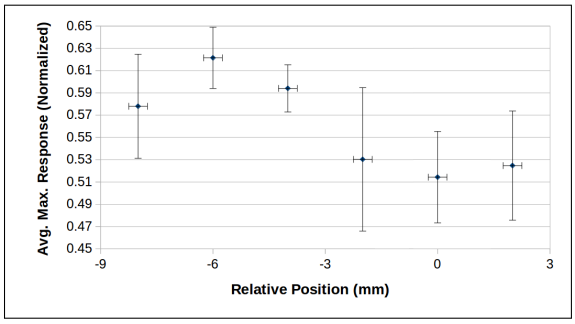
Figure 6.10: *Filled contour of the reconstructed images from the resulting raster data (32×32 grid) of a laser cross line, using COSAC's optical and acquisition subsystems and* `DMD-Raster.cpp`*, with an integration time of 1.6 ms and different RP.*

The maximum value of each raster for each relative position (RP) was then extracted and saved; for series 1-3 the rasters were performed 3 times per RP, while on the 4^{th} series they were performed 5 times for each RP; each RP's set of maximum values was then averaged and plotted (see fig. 6.11); the plot was analyzed and a smaller RP interval, corresponding to the region with a higher response, was defined for the next raster series. This process was repeated until the limit of the ruler was reached (0.5 mm).

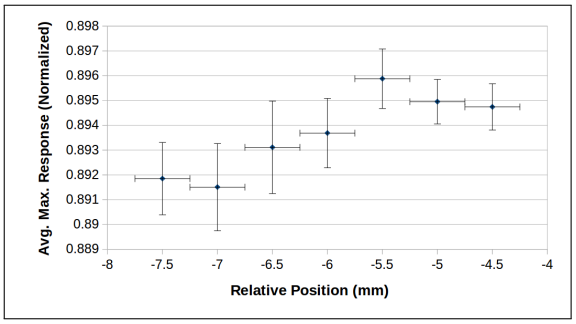
Given the standard deviation of the values presented in 6.11d, it is not possible to conclude which value maximizes the response so, their average was taken instead (-5.5 mm). The final L1-DMD focus distance was in this way determined as 149.5 mm, instead of the designed 155 mm (see fig. 6.10).



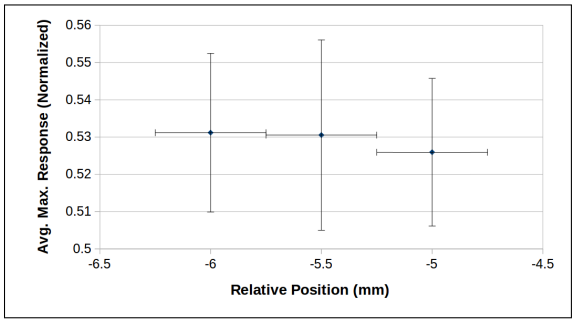
(a) *Series 1,* $IT = 400\,\mu s$.



(b) *Series 2,* $IT = 1600\,\mu s$.



(c) *Series 3,* $IT = 6400\,\mu s$.



(d) *Series 4,* $IT = 1600\,\mu s$.

Figure 6.11: *Plots of the average maximum response (normalized) acquired by the pIDDO board against the relative position of L1 (where 0 is the initial position).*

6.2.2 Punctures and cutout 'C'

For these two tests a kapaline board placed at $\sim 11\,\text{cm}$ from COSAC's aperture and a white led lantern was placed at $\sim 3\,\text{m}$ from the board (see fig. 6.12). For each test the led and board where aligned with the aperture and DMD center so that in each case the relevant structure (see fig. 6.13) was focused on the raster grid. The outcomes of the laser cross line tests were also taken into account and, as such, IT was set to 10 ms.

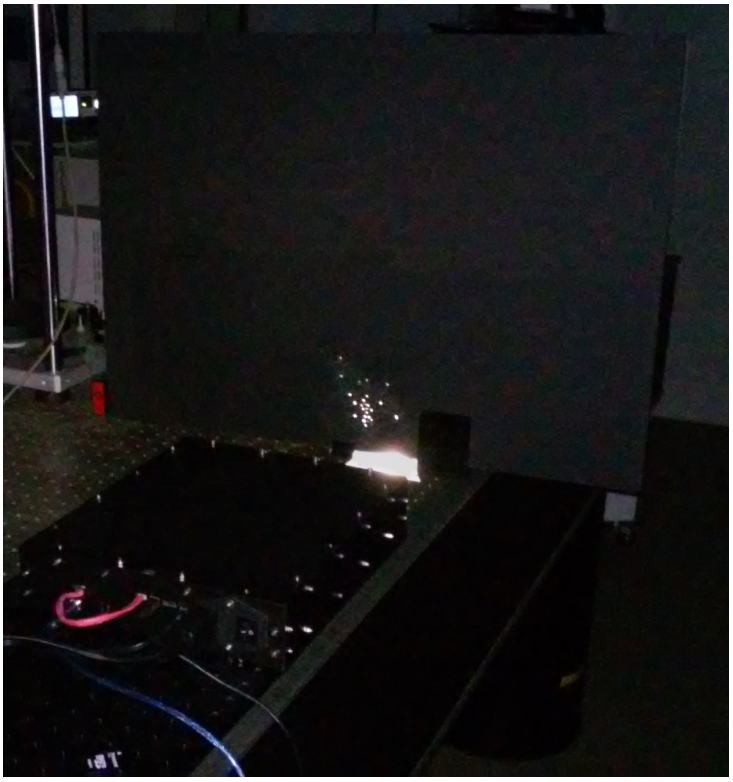
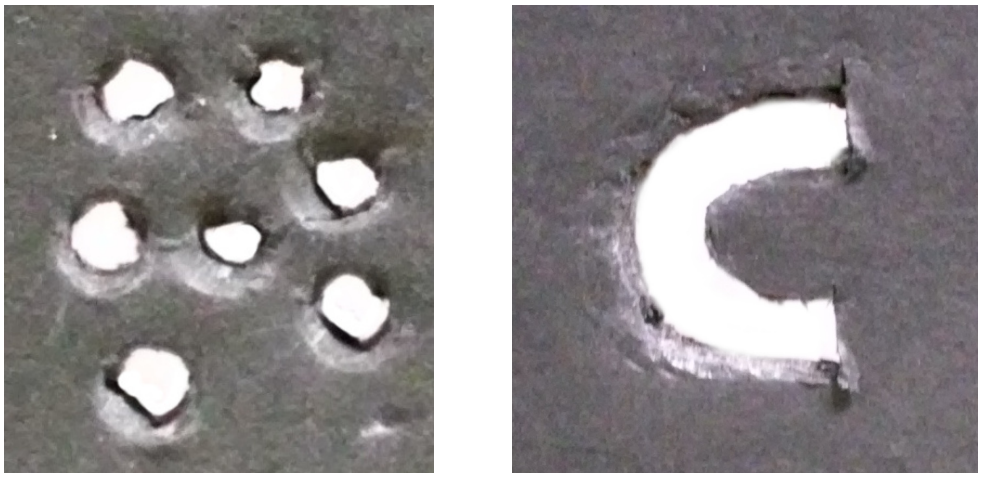


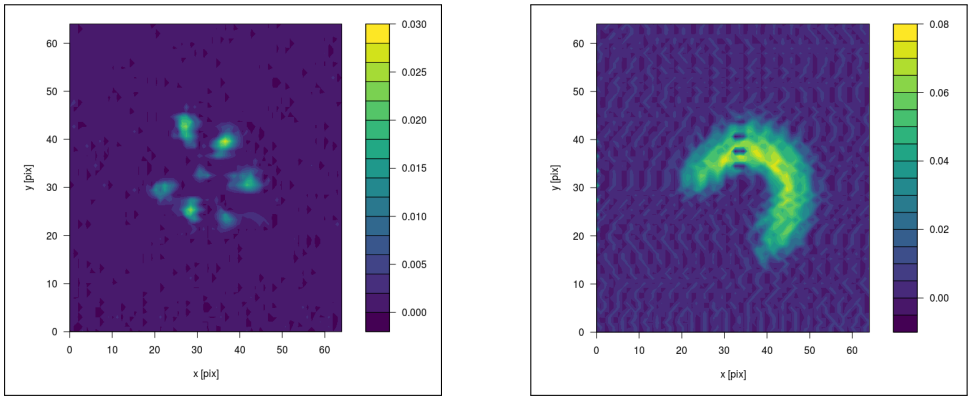
Figure 6.12: *Setup for rasterizing the projection of white light through punctures on black kapaline board. The board was placed at $\sim 11\,cm$ from COSAC's flange, while the white led lantern was placed at $\sim 3\,m$ from the back of the board.*

New rasters were performed while lowering the IT value until the structures were no longer discernible in the acquisition construction. Having set an IT lower boundary for the 32×32 raster grid of $\text{IT}_{32} = 15\,\text{ms}$, new rasters were performed with a 64×64 grid and $\text{IT}_{64} = 75\,\text{ms}$. In the results, even though it is possible to discern the shapes of the targets, it is noticeable the low magnitude of the digital values, considering they were normalized before being output; from this, came the realization of the necessity of increasing the integration time in future tests with white light. Additionally, from the results' orientation, we can recover the diamond square grid of the DMD due to the rotation applied between the input, see fig. 6.13, and the output, see fig. 6.14.



(a) *3 mm diameter punctures.* **(b)** *A cutout 'C'.*

Figure 6.13: *Structures, cutout on black kapaline, that were illuminated by a white led lantern in order to test COSAC's ability to detect mixed wavelengths of visible light.*



(a) **(b)**

Figure 6.14: *Filled contour of the reconstructed images from the resulting raster data (64×64 grid) of the projection of a white led lantern through punctures, fig. 6.14a, and a cutout 'C', fig. 6.14b, off a black kapaline board, using COSAC's optical and acquisition subsystems and* `DMD-Raster.cpp`*, with an integration time of 75 ms.*

6.3 Analysis & Comparison

Having evaluated and corrected the group behavior of COSAC's acquisition, signal coding and optical subsystems, the only thing left to perform were measurements using `DMD-CS.cpp`, instead of `DMD-Raster.cpp`, and then test the reconstruction subsystem with real measurement vectors.

During the following procedures, as well as during the calibration stage, it came to attention that the value of the first integration (this was absorved using both pIDDO-10bit and pIDDO-24bit), in almost every measurement file would be at least two times higher than the next and, more often than not, would be the maximum value present in the file. The origin of this effect remains unclear and unsolved. To bypass this issue, before processing the data, the first value of every measurement file, and the first index in the corresponding indices files, were removed.

6.3.1 Analysis

In these analyses series the same setup described in section 6.2 was used, with the previously described switch of the control program running on the PC. The laser cross line was used as a target once more and, instead of the kapaline cutouts, two images printed on acetate sheet: Faculdade de Ciências' C symbol (see fig 6.15) and the same galaxy image (see fig. 4.1) used in the simulations of section 4.1.2. To improve the contrast, these images were printed twice, cutout, and then each one was layed over its double. These overlayed prints were then glued with tape to a piece of black kapaline, with a hole slightly larger than the images, placed at the same distance as in the procedure of 6.2 (see fig. 6.16). Both versions of the acquisition circuit were used in these analyses.

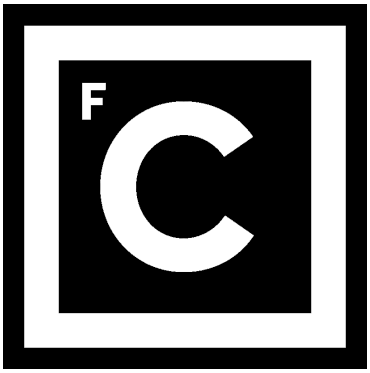


Figure 6.15: *Faculdade de Ciências da Universidade de Lisboa's symbol.*

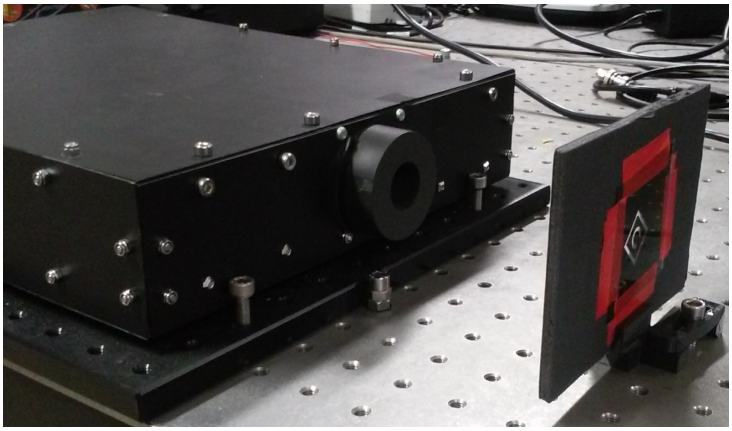


Figure 6.16: *Setup for measuring the projection of images printed on acetate. The board was placed at* ~ 11 *cm from COSAC's flange, while the white led lantern was placed at* ~ 3 *m from the back of the board. The prints were aligned with a hole on the board and glued to the board using tape.*

Studying the dependence of the signal to noise ratio with the integration time and the amount of samples performed per measurement

On the first analysis the aim was to measure the SNR's dependence on the integration time and on the amount of samples performed per measurement (as a percentage of total amount of possible

samples on the representation basis). The laser cross was used as target but a diaphragm with an apperture diameter of $\sim 2\,\text{mm}$ was coupled with COSAC's flange, to ensure enough contrast so as to easily sort signal from noise. Then, for each circuit, 16 rasters of the Hadamard matrix of rank 1024 (32×32 maps) were performed, each raster with a different integration time increasing in $50\,\mu\text{s}$ steps from $50\,\mu\text{s}$ to $800\,\mu\text{s}$.

The resulting measurement files were then randomly resampled, using a few simple instructions that were added to the code of appendix C.14, so as to perform reconstructions using only a given fraction of the whole raster. These reconstructions were realized, for each acquisition time, using 5% and from 10% to 100% in 10% steps. The codes of appendices C.14 and C.15 were changed in order to, in addition to the image file, output a text file holding the unformatted pixel values.

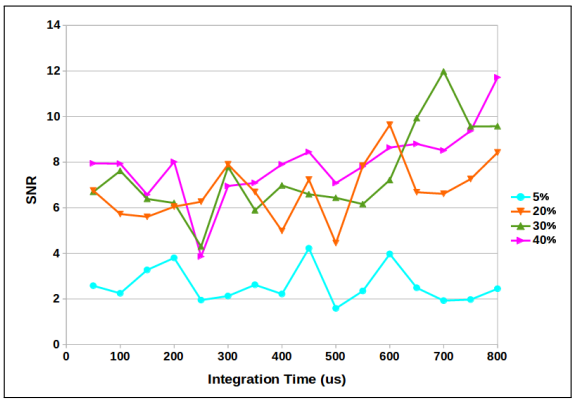
These pixel values, p, were then analysed using a matlab function written for that purpose, `IterSignalNoiseAvg.m` (see appendix C.17), that iteratively calculates their average, $\bar{p}$, and then rejects values higher than $\bar{p}+3s_p$. The iterations run until $|\bar{p'}-\bar{p}|/\bar{p} < \varepsilon$. Since the files to which this function is aimed are largely composed of background noise, this 3σ rejection procedure is effectively separating the noise (values kept in the main set) from the signal (values rejected from the main set). Prior to the iterative stage, the measurement values are normalized. The function outputs estimates for the mean values of noise and signal and their respective standard deviations.

To more efficiently process many pixel value files, another matlab script was written, `GetSNRFromFolderFiles.m`, that browses through a given folder path, parses each `txt` file in that folder through `IterSignalNoiseAvg.m`, and uses the output of that function to estimate the SNR and its uncertainty. For all `txt` files in the given folder, the file name, the estimated SNR and its uncertainty are wrote to a line of an output file.

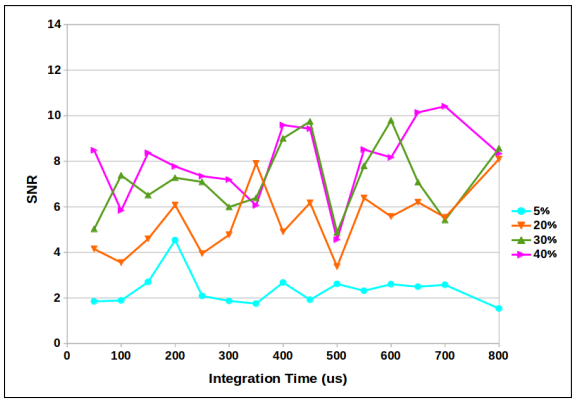
The resulting data was then sorted and plotted. In fig. 6.17, four series (for fixed percentual sample amounts of 5%, 20%, 30% and 40%) are presented expressing the relationship between the estimate SNR value (as well as its estimate uncertainty) and the integration time, for each acquisition circuit. In fig. 6.18, another four series (with fixed integration times of $100\,\mu\text{s}$, $300\,\mu\text{s}$, $600\,\mu\text{s}$ and $800\,\mu\text{s}$) are presented expressing the relationship between the estimate SNR value (as well as its estimate uncertainty) and the percentual ABV, for each acquisition circuit.

As it was observed during the simulation stage, the results show that the SNR increases with the ABV. But, after a given percentage ($\sim 40\,\%$), that increase slows down and the SNR eventually appears to converge, while displaying some oscillations, for the 10 bit version, while for the 24 bit version, with larger IT values, the SNR appears to keep increasing. These same oscillations are also present in the relationship between the SNR and the IT, dispite them however, a positive dependence can be infered (which could have been intuitively expected) in three out of the four series displayed.

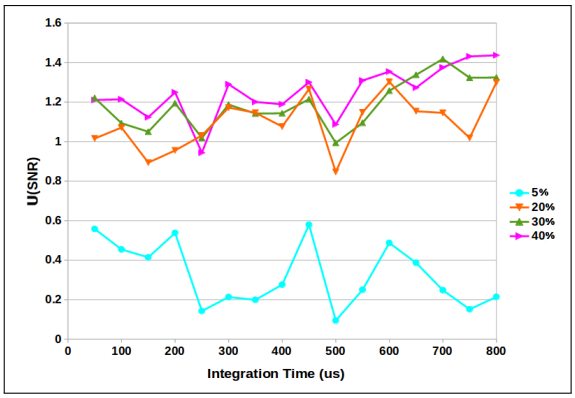
These results are consistent with the mathematical formulation of the problem, since the more projections of signal the more likely it is to increase the amount of information available for the reconstruction, but the quality of the reconstruction is dependent on relative quality of the vectors upon which the signals are projected, and not only do we randomly sample the available vectors we also do not know which would be best fitted should we want to chose them. As an example of the impact vector quality has on reconstruction, in fig. 6.19 three reconstructions using three different resamples, of the same size ($5\,\%$), from the same measurument series (pIDDO-10bit, IT=$250\,\mu\text{s}$) are presented.



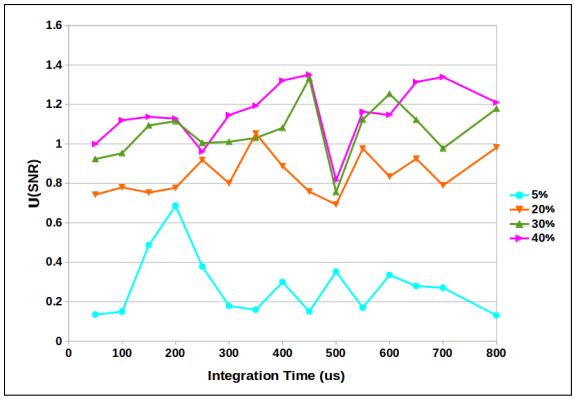
(a) *SNR vs IT - pIDDO-10bit.*



(b) *SNR vs IT - pIDDO-24bit.*



(c) U_{SNR} *vs IT - pIDDO-10bit.*



(d) U_{SNR} *vs IT - pIDDO-24bit.*

Figure 6.17: *Plots of SNR estimates (figs. 6.17a and 6.17b), and their estimated uncertainties (figs. 6.17c and 6.17d), against the integration time (expressed in µs), fixating the percentual amount of samples used.*

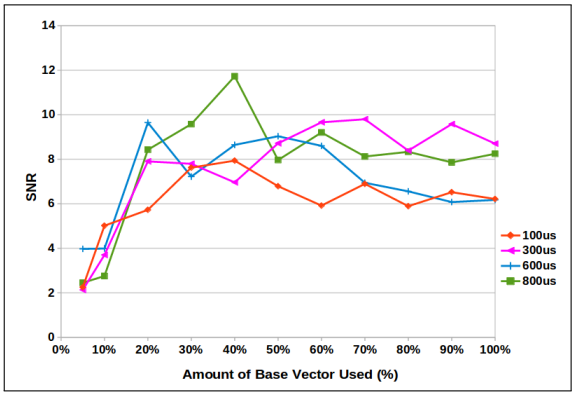
Attempting to correct measurements

On the second analysis we wished to measure the impact of dark and flatfield corrections on measurements. The two prints were used as targets and the diaphragm was removed. For each circuit, one dark (COSAC's apperture was covered and the led lantern was turned off) and one flatfield (no obstruction between the apperture and led lantern) measurements were performed, rasterizing an Hadamard base of rank 4096 (64×64 maps) with an integration time of 20 ms. This same measurement procedure was realized with both prints.

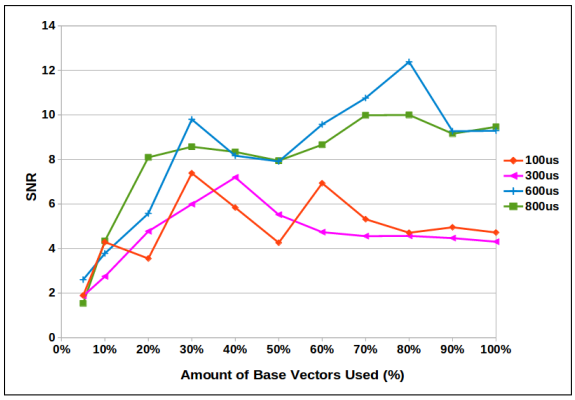
For each file resulting from the prints' measurements, five CS reconstructions were made (with ABV=10 %)[3]; using the `IterSignalNoiseAvg.m` function, an average value and standard deviation for the SNR estimate were infered. Afterwards the same prodecure was repeated but `ReconstructImageFromFiles.m` was edited to instead of simply resampling the measurement files, y_i, sampling the result of correcting the measurement file values, y_i*, with the values of the dark and flatfield measurements, dv_i and fv_i respectively, via eq. 6.1, where the fv_i were previously normalized.

$$y_i* = \frac{y_i - dv_i}{fv_i} \tag{6.1}$$

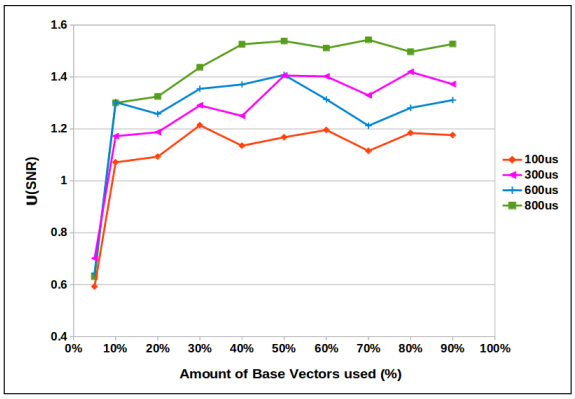
[3]It was decided to apply this correction method to ABV=10 % CS reconstructions so that improvements or deterioration of the reconstructions could be easily visually detected, which wouldn't be so for ABV>30 %.



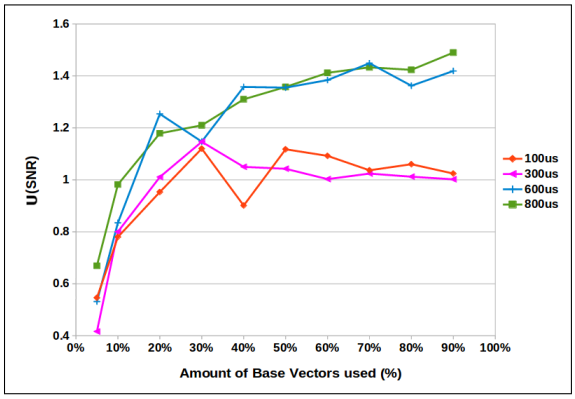
(a) *SNR vs ABV - pIDDO-10bit.*



(b) *SNR vs ABV - pIDDO-24bit.*



(c) U_{SNR} *vs ABV - pIDDO-10bit.*



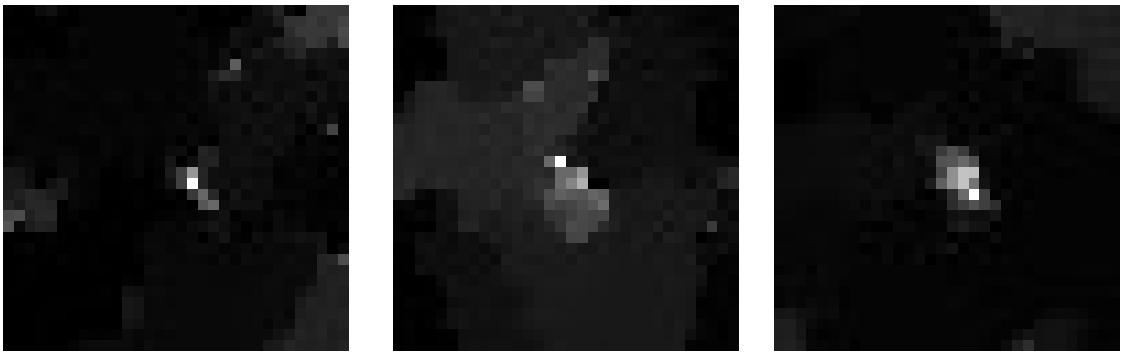
(d) U_{SNR} *vs ABV - pIDDO-24bit.*

Figure 6.18: *Plots of SNR estimates (figs. 6.18a and 6.18b), and their estimated uncertainties (figs. 6.18c and 6.18d), against the integration time (expressed in µs), fixating the percentual amount of samples used.*

The results were unclear, as can be seen in tab. 6.1, in most cases the average SNR estimate decreased, this was not expected. The proportionally larger decrease in the standard deviation could suggest improved image contrast nevertheless.

To help analyse these results, the average of each CS reconstruction series was computed, normalized and saved as an image file. By analysing the image files, however, the effectivess of the correcting procedure was proven flawed if not detrimental (see fig. 6.20), with only the pIDDO-24bit measurument of the C showing a slight contrast improvement having been corrected, which puts into question whether the improvement, in that case, occurs because or dispite of the correcting procedure. The remaining measurements show no improvement or even deterioration, suggesting that the flatfield is introducing noise into the measurements (which can clearly be seen in the black spot of figs. 6.20b and 6.20d). Considering what was just described, this correction procedure wasn't applied to anyother measurement.

A possible cause for the failure of the aforementioned method is the fact that it disregards the mismatch between the shape of COSAC's apperture and the section of the DMD in charge of the signal encoding, as well as the spot of the focused signal relative to that same section's area.

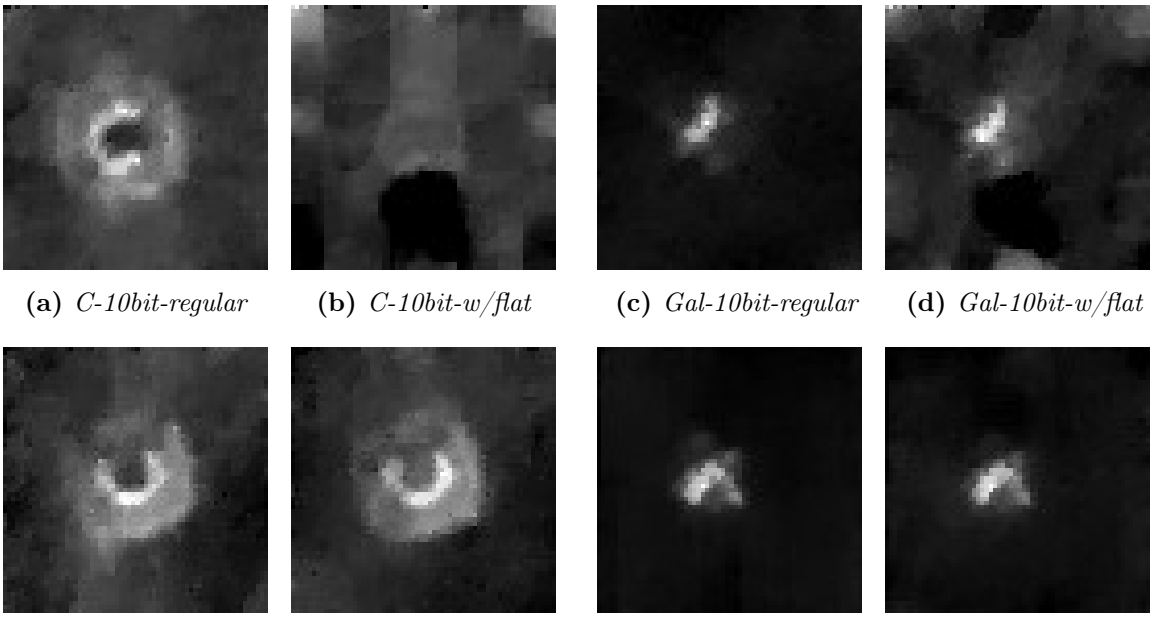


(a) **(b)** **(c)**

Figure 6.19: *Three CS reconstructions using different sets of 5 % of the total amount of possible samples of scene (laser cross obstructed by a diaphragm). The measurement file was produced using pIDDO-10bit, IT=250 µs, and the Hadamard matrix of rank 1024.*

Print	24 bit				10 bit			
	Regular		Corrected		Regular		Corrected	
	E(SNR)	s_{SNR}	*E(SNR)*	s_{SNR}	*E(SNR)*	s_{SNR}	*E(SNR)*	s_{SNR}
C	1.649	0.351	1.325	0.149	1.763	0.362	1.767	0.301
Galaxy	1.884	0.023	1.314	0.072	2.280	0.507	2.097	0.240

Table 6.1: *Average and standard deviation for estimated SNR values of CS reconstructions of regular and corrected measurement samples. The targets measured were black and white prints on acetate sheet, one of Faculdade de Ciências' symbol, C, the other of the galaxy of fig. 4.1; the measurements were performed using the Hadamard matrix of rank 4096 and IT=20 ms.*



(a) *C-10bit-regular* **(b)** *C-10bit-w/flat* **(c)** *Gal-10bit-regular* **(d)** *Gal-10bit-w/flat*

(e) *C-24bit-regular* **(f)** *C-24bit-w/flat* **(g)** *Gal-24bit-regular* **(h)** *Gal-24bit-w/flat*

Figure 6.20: *Average images of the CS reconstructions series, analysed in tab. 6.1, for the C symbol and the Galaxy (Gal) prints.*

6.3.2 Visual Comparison

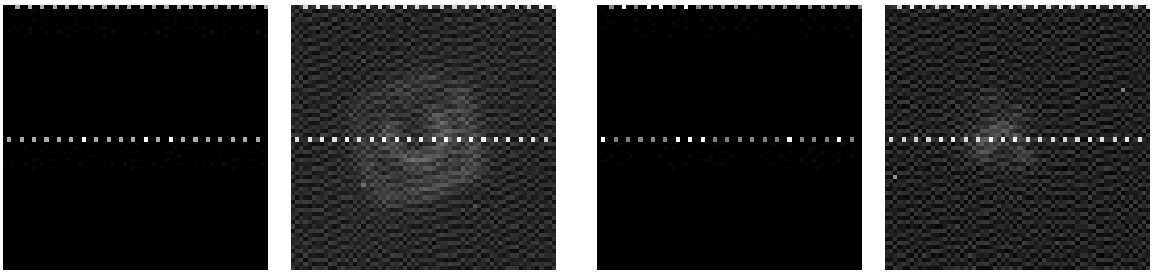
Taking into account the results of the previous section, a performance comparison was made between measuring the acetate prints through pixel raster, Hadamard base raster (Hadamard transform optics), and CS reconstructions with different ABV percentages, for each pIDDO version.

All measurements were performed using 64×64 maps; the CS measurements were performed at IT=20 ms; for each comparison the integration time of the other measurement methods was fixed so as to equalize the total integation time - total acquisition time minus the overhead -, the integration time of each measurement can be consulted in tab. 6.2.

CS			**Total IT (s)**	**Pixel/Hadamard Raster**	
ABV (%)	**K**	**IT (s)**		**K**	**IT (s)**
40	1638	0.02	32.76	4096	0.008
30	1229		24.58		0.006
20	819		16.38		0.004
10	410		8.2		0.002

Table 6.2: *Settings used for each measurement method (CS, pixel raster and Hadamard base raster) in order to properly compare the estimated SNR values for the same total integration time (note that even though both rasters will perform the same amount of samples, they'll do so in different bases).*

The yielded results were somewhat surprising. It was expected that, due to the reduced ratio of integration time to signal reflected into the sensor, the pixel raster resulting images would be faded, these however were simply black with the corrresponding data file presenting an estimate SNR inferior to 1. An extra raster measurement was then performed for each print and pIDDO board combination, with IT=20 ms, to both compare the resolution of both boards and provide a visual reference for both Hadamard and CS reconstructions (see fig. 6.21). In all these rasters two noise lines can be noticed, their origin, their source however remains unclear and the effect doesn't manifest itself in the two other measurement modes.



(a) *C-10bit* **(b)** *C-24bit* **(c)** *Galaxy-10bit* **(d)** *Galaxy-24bit*

Figure 6.21: *Pixel raster results for measuruments of both prints, using both versions of the acquisition circuit (10 bit and 24 bit), with an integration time of 20 ms.*

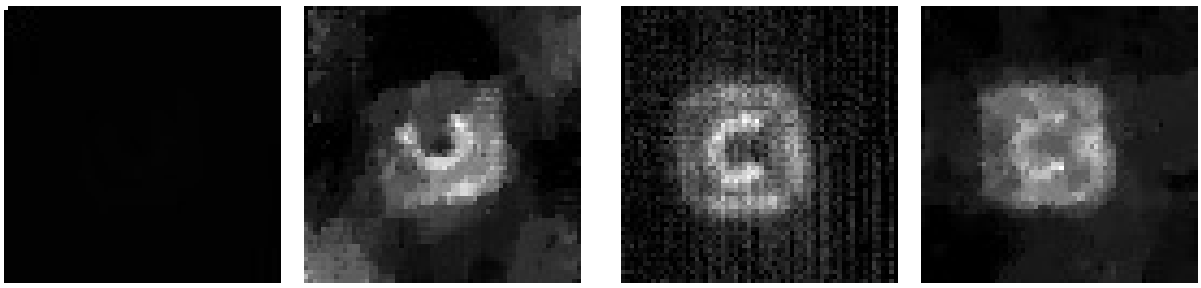
More surprising were the results of the Hadamard base raster and its reconstruction. Because it is a well determined system, reconstructing the signal x, in this case, involves simply resolving the following equation:

$$x = A^{-1}y \tag{6.2}$$

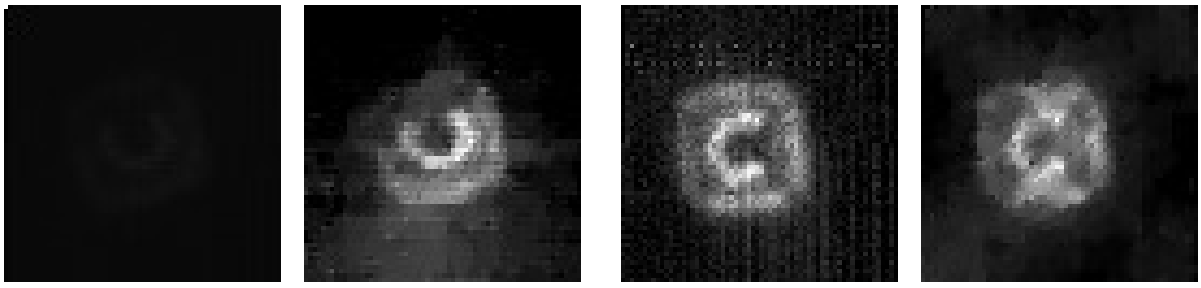
We note, however, that only two reconstructions were successful adopting this approach. The other reconstructions resulted in black images with a single white pixel. Analysing the corresponding data files revealed that for each of the failed reconstructions, only one pixel value was positive and 3 to 4 magnitude orders larger than the rest. This may be indicative of the direct inversion method (eq. 6.2) oversensitivity to noise.

In fig. 6.22 the successful Hadamard raster reconstructions are compared against the equivalente CS reconstructions (see tab. 6.2), an example of failed Hadamard reconstruction is also displayed. For IT=4 ms the successful Hadamard raster is clearly sharper than the corresponding CS measurements, for here we can not only recover most of the targets' structure but also structures resulting from the overlap of the doubled prints (figs. 6.22c and 6.22g), while the 10 bit-20% reconstruction is smudged. The quality of the CS reconstructions increase slightly with the increase of ABV from 20% to 30%, with the 24 bit result starting to display a similar granular structure; the correspondent Hadamard raster however displays more noise polution with the IT increase. One last thing to notice from these images is the fact that even though the contrast of both 10 bit and 24 bit CS measurements is similar, the target structure is sharper on the 24 bit reconstruction.

In fig. 6.23 the remaining CS results obtained with both pIDDO-10bit and pIDDO-24bit can be compared.



(a) *Had-IT=4 ms-24bit* **(b)** *CS-ABV=20%-24bit* **(c)** *Had-IT=4 ms-10bit* **(d)** *CS-ABV=20%-10bit*

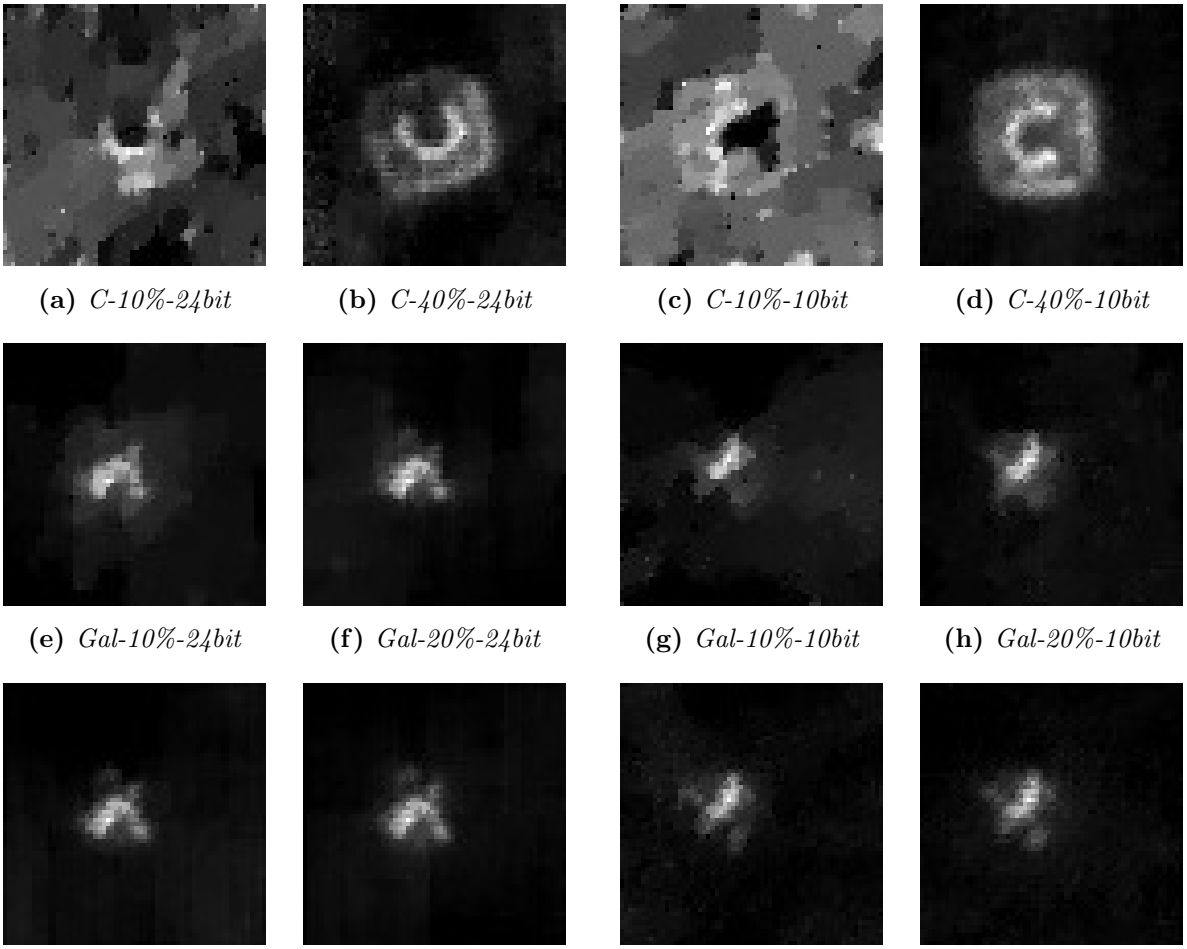


(e) *Had-IT=6 ms-24bit* **(f)** *CS-ABV=30%-24bit* **(g)** *Had-IT=6 ms-10bit* **(h)** *CS-ABV=30%-10bit*

Figure 6.22: *Comparison between equivalent reconstructions (see tab. 6.2) of Hadamard rasters (Had) and CS measurements of the C symbol, performed with both pIDDO versions (10 bit and 24 bit).*

Considering the COSAC's satisfactory performance, at reconstructing the main structure of both the C symbol and the galaxy prints, using the rank 4096 Hadamard matrix, it was attempted to better resolve the structure of the C symbol by making some measurements using the Hadamard matrix of rank 16384.

By analysing the resulting data files, the acquisition stage appears to have been successful. The reconstruction stage, however, was not, with the computational process being terminated due to the memory required by it not being available.



(a) *C-10%-24bit* **(b)** *C-40%-24bit* **(c)** *C-10%-10bit* **(d)** *C-40%-10bit*

(e) *Gal-10%-24bit* **(f)** *Gal-20%-24bit* **(g)** *Gal-10%-10bit* **(h)** *Gal-20%-10bit*

(i) *Gal-30%-24bit* **(j)** *Gal-40%-24bit* **(k)** *Gal-30%-10bit* **(l)** *Gal-40%-10bit*

Figure 6.23: *Comparison between reconstructions of CS measurements of the C symbol and the Galaxy (Gal), performed with both pIDDO versions (10 bit and 24 bit), using different ABV values. The images are labeled in the following scheme target-ABV-pIDDO version.*

Chapter 7

Conclusions & Future Applications

In this work we've shown that a compressed sensing based imaging device for astronomy is in fact feasible. Not only were a sparse coding, a reconstruction and an optical subsystems studied and developed, as they were successfully proved to be functional and integrated within a fully functional and selfcontained prototype for COSAC: a Compressed Sensing Astronomical Camera.

The acquisition subsystem initially developed by Bandarra[20] was improved and branched into to two functional versions with different resolutions (10 bit and 24 bit), moreover the performance of the 10 bit acquisition circuit exceeded expectations, showing that the adequacy of the chosen coding basis is more important than the measurement's resolution. Additionally, a casing and structural subsystems have been designed, developed and assembled, enabling the coupling of COSAC to standard equatorial telescopic mounts. It wasn't yet possible yet to perform sky measurements, as we lack a supporting structure to enable a safe coupling between COSAC and a telescope. This is planned, however, to take place in the near future. A fully functioning COSAC was achieved, though it presents some limitations commented bellow.

The acquisition subsystem isn't, currently, as sensitive as it should be for the intended application, requiring a minimum integration time of 10 ms so that a white light scene can be recognizable from the reconstructed measurement, when the light source has a 8 mm apperture and is 3 m away from the optical subsystem. Part of this limitation can be lifted, however, by disconnecting some of the internal capacitors of the IVC, which were all connected to avoid easily saturating its output, therefore facilitating the study of the acquisition circuit.

The signal coding subsystem is limited on two fronts: first, the fact that the communication between the PC and the DM365 is peformed via USB 1.0 establishes a $\sim$7 s overhead between each sequence of 96 coding masks, but even if this wasn't the case it would be better to generate the coding maps directly inside an FPGA near the DMD, thus cutting the entire middleman cirtuitry (DM365); secondly, at the present moment only Hadamard matrices can be used as coding bases by the implemented software, but this can be changed as easily and replaced by more optimized coding bases – this will certainly improve the results, enabling COSAC to perform more accurate measurements with lower ABV, as it has been demonstrated possible. The use of matrices where the signal's sparsity is higher will also justify the use of sparse representation of observation vectors and coding matrices, thus cutting down also the computational memory cost of the reconstruction algorithms.

The casing and supports for the optical subsystem can also be improved, to better isolate the hardware from the instruments' surroundings, and to facilitate the alignment of the optical components, respectively. A possible new iteration of COSAC could also include a Raspberry

Pi and an interactive display integrated in the casing, the Raspberry could replace the Arduino and even eliminate the need of an outside control computer.

The present supporting structure for COSAC allows for the addition of a second sensor. This could be added, along with another set of IC's, to the pIDDO board, with the control Arduino pins being shared for simultaneous measurements, either on different wavelength passband, to produce a dual wavelength image, or on the same wavelength, thus recovering the entire flux of the signal, which would be useful for monitoring the stability of the system or undesirable external variations of the light level. We can also consider an operation using polarizers in the two channels, and thus recover the polarization of the light in the reconstructed signal.

COSAC is a materialized proof of concept for the development of compressed sensing astronomical instruments. As it was implemented in this work, is has not a large interest for astronomical observations as the present photodiode is mostly sensitive to optical wavelengths, and we known how to make inexpensive spatially resolved arrays with large number of pixels in these wavelengths. However, by simply adopting another photodiode and perhaps a different coating in the DMD mirrors, COSAC can be easily modified to operate in wavelengths where either the spatially resolved detectors are very expensive (e.g. $\sim 1 - 2.5\mu$m) or where no high spatial resolution detector is readily available in the market (e.g. $\gg 2.5\mu$m). Another direct potential application of COSAC is the development of an integral field unit instrument using the same technology basis as COSAC, but replacing the photodiode by a CCD and adding a dispersive element in the optical path. Such development would then enable high spatial resolution IFUs to be build in very small volumes, advancing the state of such instruments in ground based telescopes, and also enabling these instruments to fly in satellites: something that is presently unfeasible using the current technology. COSAC itself already hints to space: its design could fit in a 6 units cubesat, it could be further optimized to fit in even tigher volumes, and it could easily enable IFUs capable of reconstructing datacubes of 256×256 spectra to be entirely built within shoebox-sized enclosures using readily available technologies.

References

[1] D. Donoho. Compressed sensing. *IEEE Trans. Inf. Theory*, 52(4), Sep. 2006.

[2] E. Candès. Compressive sampling. *Proc. Int. Congr. Math.*, 3, 2006.

[3] E. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, Vol. 52, Issue 2:489-509, Feb. 2006.

[4] E. Candès and T. Tao. Near-optimal signal recovery from random projections and universal encoding strategies. *IEEE Transactions on Information Theory*, Vol. 52, Issue 12:5406-5425, Jan. 2007.

[5] R. Baraniuk. Compressive sampling. *IEEE Signal Process. Mag.*, 24(4), Jul. 2007.

[6] E. Candès and M. Wakin. An introduction to compressive sampling. *IEEE Signal Process. Mag.*, 25(2), Mar. 2008.

[7] D. Mackenzie. Compressed Sensing Makes Every Pixel Count. *What's Happening in the Mathematical Sciences, Vol. 7*, AMS, pp. 114-127, 2009. ISBN 978-0821844786.

[8] M. Duarte and Y. Eldar. Structured compressed sensing: From theory to applications. *IEEE Transactions on Signal Processing*, 59, Sep. 2011.

[9] D. Takhar, J. Laska, M. Wakin, M. Duarte, D. Baron, S. Sarvotham, K. Kelly, and R. Baraniuk. A new compressive imaging camera architecture using optical-domain compression. *Proc. Computational Imaging*, IV, 2013.

[10] D. Brady, K. Choi, D. Marks, R. Horisaki and S. Lim. Compressive holography. *Optics Express*, 17: 13040–1304, 2009.

[11] Y. Rivenson, A. Stern, and B. Javidi. Compressive Fresnel Holography. *Journal of Display Technology*. 6(10), 506–509, 2010.

[12] M. Lustig, D. L. Donoho and J. M. Pauly. Sparse MRI: The application of compressed sensing for rapid MR imaging. *Magnetic Resonance in Medicine*, 58(6), 1182–1195. 2007.

[13] M. Lustig, D. L. Donoho, J. M. Santos and J. M. Pauly. Compressed Sensing MRI. *IEEE Signal Processing Magazine*, 25(2), 72–82. 2008.

[14] J. Bobin, J. Starck, and R. Ottensamer. Compressed sensing in astronomy. *IEEE Journal of Selected Topics in Signal Processing*, 2(5), Oct. 2008.

[15] Y. Oike and A. El Gamal. CMOS image sensor with per-column ΣΔ ADC and programmable compressed sensing. *IEEE Journal of Solid-State Circuits*, 48(1), Jan. 2013.

[16] W. Chan, K. Charan, D. Takhar, K. Kelly, R. Baraniuk, and D. Mittleman. A single-pixel terahertz system based on compressed sensing. *Applied Physics Letters*, 93, 2008.

[17] H. Huang, S. Misra, W. Tang, H. Barani and H. Al-Azzawi. Applications of Compressed Sensing in Communications Networks. *arXiv:1305.3002v3*, 5 Feb. 2014.

[18] A. Stevens, L. Kovarik, P. Abellan, X. Yuan, L. Carin and N. D. Browning. Applying compressive sensing to TEM video: a substantial frame rate increase on any camera. *Advanced Structural and Chemical Imaging*, 1(1), 2015.

[19] Optical compressive sensing technologies for space applications – CNR-IFAC. *https://www.esa.int/spaceinvideos/Videos/2018/07/Optical_compressive_sensing_technologies_for_spac IFAC*. Last updated 9 July 2018.

[20] R. Bandarra. Electronics Design and Implementation of a Compressed Sensing Instrument for Astronomy. Dissertation, Universidade de Lisboa, 2015. Retrieved from *http://hdl.handle.net/10451/22418*.

[21] J. Pires. Implementação de um sistema Óptico e Optomecânico para um instrumento astronómico baseado no Compressed Sensing. Dissertation, Universidade de Lisboa, 2016. Retrieved from *http://hdl.handle.net/10451/25567*.

[22] L. Hornbeck. Digital Micromirror Device. *US Patent No. 5061049*, Inducted in 2009.

[23] DLPLIGHTCRAFTER DLP LightCrafter Evalution Module | TI.com. *http://www.ti.com/tool/DLPLIGHTCRAFTER*. Last visited Dec. 2018.

[24] J. Hadamard. Résolution d'une question relative aux déterminants. *Bulletin des Sciences Mathématiques*, 17: 240–246, 1893.

[25] J.J. Sylvester. Thoughts on inverse orthogonal matrices, simultaneous sign successions, and tessellated pavements in two or more colours, with applications to Newton's rule, ornamental tile-work, and the theory of numbers. *Philosophical Magazine*, 34:461–475, 1867.

[26] Stanford. L1-Magic. *https://statweb.stanford.edu/ candes/l1magic/*. Last visited 2018-07-11.

[27] Wikipedia. *https://en.wikipedia.org/wiki/BMP_file_format#Example_1*. Last visited Oct. 2017.

[28] H. Nyquist. Certain topics in telegraph transmission theory. *Trans. Amer. Inst. Electr.Eng.*, 47, Apr. 1928.

[29] C. E. Shannon. Communications in the presence of noise. *Proc. IRE*, 37, Jan. 1949.

[30] M. Harwit, N. J. A. Sloane. Hadamard Transform Optics.*Academic Press*, 1979.

[31] D. Schneider. New Camera Chip Captures Only What It Needs. *IEEE Spectrum*, March 2013.

[32] Hamamatsu. Si PIN photodiodes S1223 series For visible to near IR, precision photometry. Feb. 2013.

[33] Burr-Brown. Precision Switched Integrator Transimpedance Amplifier, 1996.

[34] Linear Technology. LTC2440 24-Bit High Speed Differential $\Delta\Sigma$ ADC with Selectable Speed/Resolution, 2002.

[35] Arduino. *https://www.arduino.cc/en/Reference/Wire*. Last visited 2018-02-13.

[36] Adafruit. Micro SD Card Breakout Board Tutorial. Last updated on 2017-12-11 11:05:59 PM UTC.

[37] Texas Instruments. LM1086 1.5-A Low Dropout Positive Regulators. Revised April 2015.

[38] Basics of the SPI Communication Protocol. *http://www.circuitbasics.com/basics-of-the-spi-communication-protocol/*. Last visited on 2018-05-11.

[39] Dangerous Prototypes. Interface between Arduino DM board and Linear Tech LTC2440 24-bit ADC. *http://dangerousprototypes.com/forum/viewtopic.php?t=4247*. Last updated Nov. 12 2010.

[40] R. J. Turyn. Sequences with small correlation. *H. B. Mann (ed.). Error Correcting Codes*. Wiley. pp. 195–228, 1969.

[41] P. J. Mccarthy. Pseudo-replication: Half samples. *Review of the International Statistical Institute*, 37 (3), 239-264. 1969.

[42] Z. E. Russell. Coded Aperture Magnetic Sector Mass Spectrometry. Dissertation, Duke University. Retrieved from *http://hdl.handle.net/10161/11396*.

[43] Texas Instruments. DLP LightCrafter Evaluation Module - User's Guide. Revised Nov. 2014.

[44] Texas Instruments. DLP3000 DLP® 0.3 WVGA Series 220 DMD. Revised March 2015.

[45] Texas Instruments. DLP LightCrafter DM365 Command Interface Guide. Revised March 2013.

[46] Texas Instruments. DLPC300 DLP Digital Controller for the DLP3000 DMD. Revised Aug. 2015.

[47] Texas Instruments. DLPC300 Programmer's Guide - User's Guide. Revised July 2013.

[48] [Resolved] DLPC300: Aspect Ratio and Distortion - DLP® Products Forum - DLP® Products - TI E2E Community. *https://e2e.ti.com/support/dlp/f/94/t/624074*. Last visited Sept. 2017.

[49] Teuniz. RS-232 for Linux and Windows. *https://www.teuniz.net/RS-232/*. Last updated Nov. 22 2017.

[50] Rigol. DS6000 series Digital Oscilloscope. 2011.

[51] Thorlabs. PDA100A(-EC) Si Switchable Gain Detector - User Guide. Revised June 2017.

[52] Korad. Digital-Control and Programmable DC Power Supply KA3000-6000 Series User Manual.

[53] S. Boyd and L. Vandenberghe. Convex Optimization. *Cambridge University Press*. 2004.

[54] E. candès and J. Romberg. ℓ_1-MAGIC: Recovery of Sparse Signals via Convex Programming. *Caltech*. Oct. 2005.

[55] S. S. Chen, D. L. Donoho and M. A. Saunders. Atomic Decomposition by Basis Pursuit. *SIAM Journal on Scientific Computing*, Volume 20, Number 1, 1998, pages 33–61. Feb. 2001.

[56] S. Kim, K. Koh, M. Lustig, S. Boyd and D. Gorinevsky. An Interior-Point Method for Large-Scale ℓ_1-Regularized Least Squares. *IEEE Journal of Selected Topics in Signal Processing*, Vol. 1, No. 4, Dec. 2007.

[57] M. Figueiredo, R. Nowak and S. Wright. Gadient Projectio for Sparse Reconstruction: Application to Compressed Sensing and Other Inverse Problems. *IEEE Journal of Selected Topics in Signal Processing*, Vol. 1, No. 4, Dec. 2007.

[58] GPSR. *http://www.lx.it.pt/ mtf/GPSR/*. Last visited 2018-12-01.

[59] J. Renegar. A mathematical view of interior-point methods in convex optimization. MPS-SIAM Series on Optimization. *SIAM*, 2001.

[60] S. J. Wright. Primal-Dual Interior-Point Methods. *SIAM Publications*, 1997.

[61] Eigen. *http://eigen.tuxfamily.org/*. Last updated 28 August 2018.

[62] 300mm, linear slide rail and carriage - MakerBeam. *https://www.makerbeam.com/300mm-linear-slide-rail-and-carriage.html*. Last visited 2018-09-18.

[63] OpenBeam - 15×15mm aluminum profile - MakerBeam. *https://www.makerbeam.com/openbeam/*. Last visited 2018-09-18.

[64] Corner cubes black (12p) 15mm×15mm×15mm - MakerBeam. *https://www.makerbeam.com/openbeam-corner-cubes-black-12p-for-openbeam.html*. Last visited 2018-09-18.

[65] D. L. Logan. A first course in the finite element method. *Cengage Learning*, 2011. ISBN 978-0495668251.

[66] GrabCAD. Arduino UNO Reference Design. *https://grabcad.com/library/arduino-uno-reference-design*. Last updated 2011.

Appendices

Appendix A

pIDDO

A.1 pIDDO-10bit

In this appendix we present the schematic, printed circuit board design and part list for the implementation of the 10 bit version of the acquisition circuit, pIDDO-10bit.

A.1.1 Part List

In this section we present a list of parts, with their board labels, used in the production of the pIDDO-10bit v1.1 circuit.

- 0.1 μF Capacitors: C_2, C_4
- 10 μF Capacitors: C_1, C_3
- 5 kΩ Trimmer Potentiometer: POT
- Arduino Uno Rev. 3: ARDUINO
- Arduino Stackable Header Kit
- Hamamatsu S1223 - Si PIN photodiode: PD
- IVC102 - Precision Switched Integrator Transimpedance Amplifier: IVC102
- Micro SD Card Breakout Board: SDCARD
- Raster Signal wire-board 1.25mm, 4 pin Socket: RASTER

A.1.2 Schematics

In this section we present the schematic for the 10 bit version of the acquisition circuit, pIDDO-10bit v1.1.

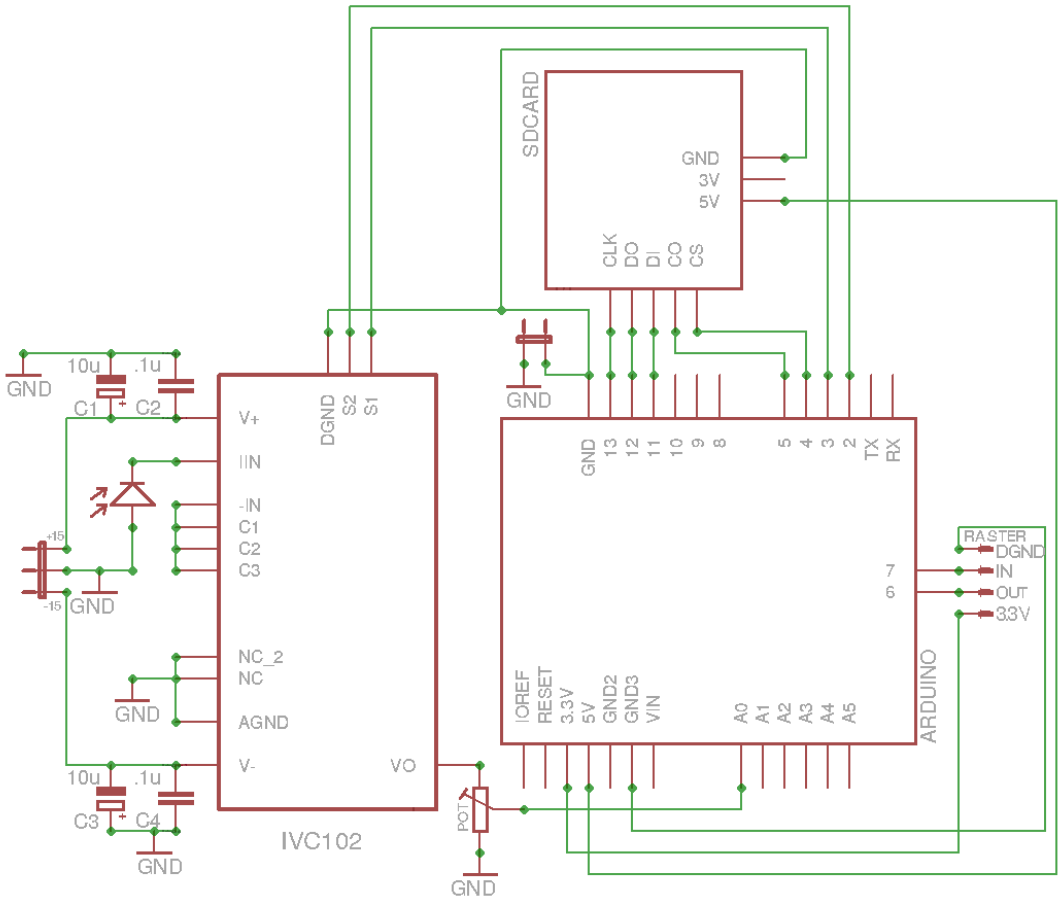


Figure A.1: *pIDDO-10bit v1.1 schematic.*

A.1.3 PCB

In this section we present the bottom and top layers of the PCB for the 10 bit version of the acquisition circuit, pIDDO-10bit v1.1.

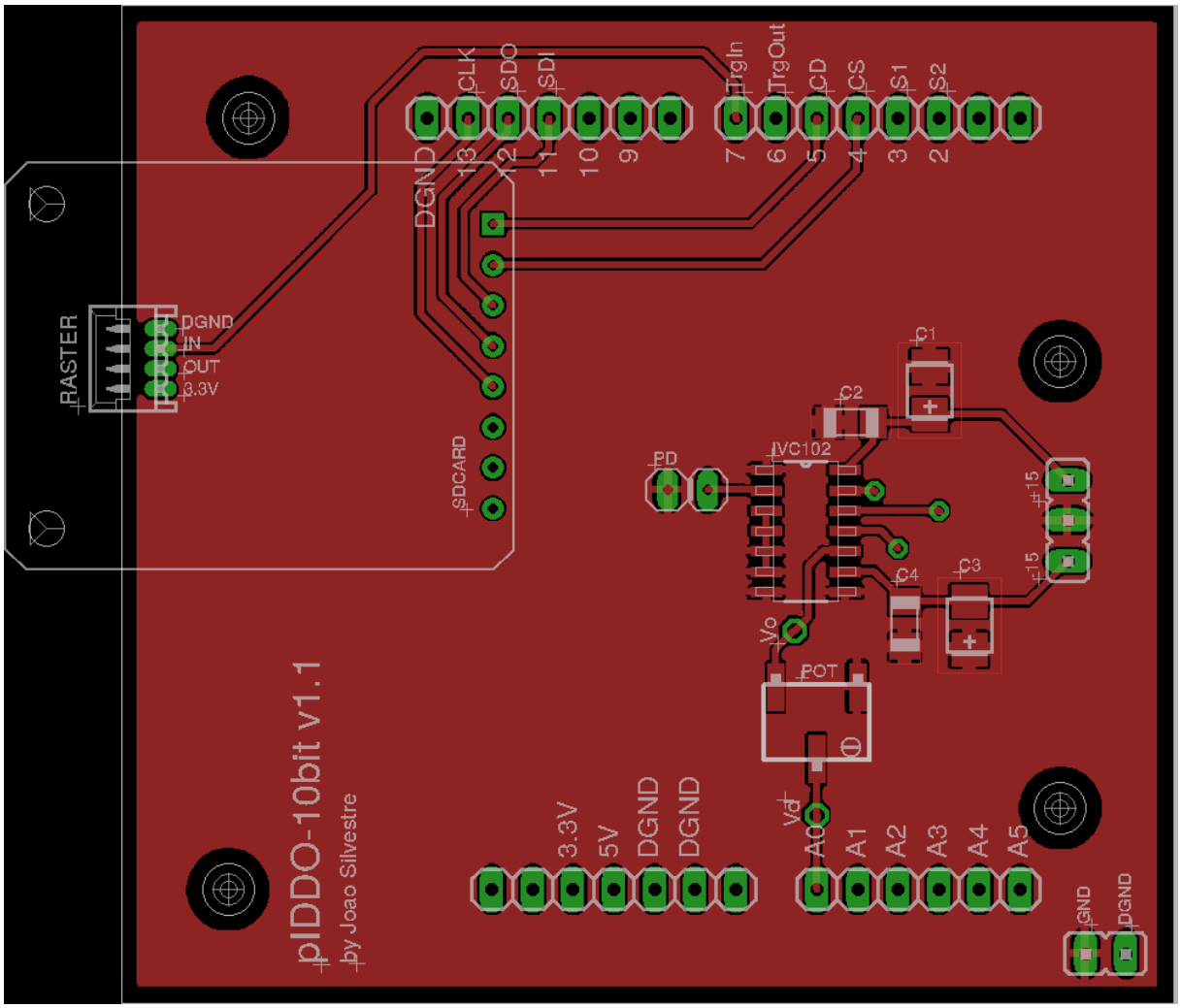


Figure A.2: *pIDDO-10bit v1.1 PCB top layer.*

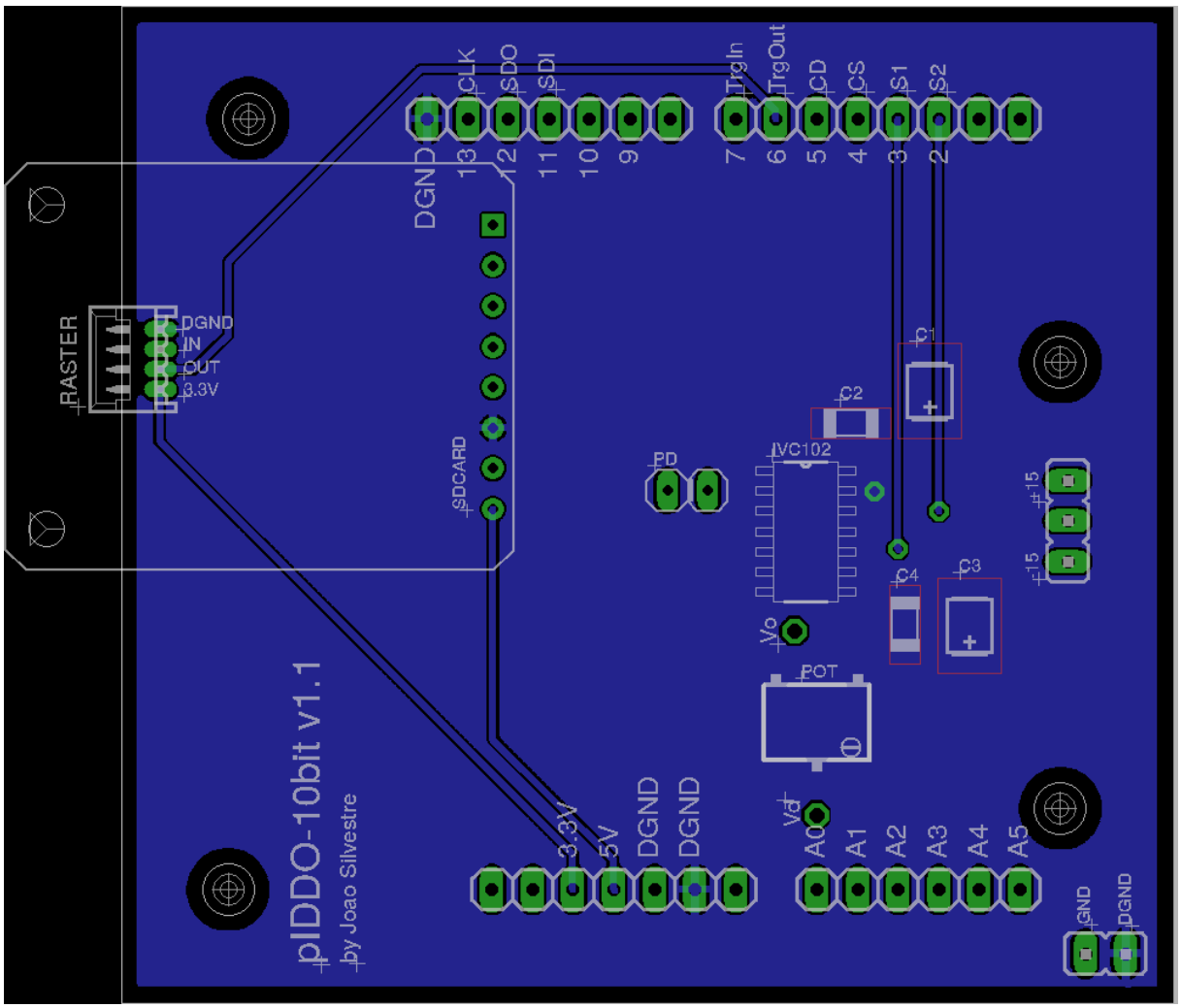


Figure A.3: *pIDDO-10bit v1.1 PCB bottom layer.*

A.2 pIDDO-24bit

In this appendix we present the schematic, printed circuit board design and part list for the implementation of the 24 bit version of the acquisition circuit, pIDDO-24bit.

A.2.1 Part List

In this section we present a list of parts, with their board labels, used in the production of the pIDDO-24bit v1.6 circuit.

- 0.1 μF Capacitors: C_3, C_5, C_7
- 10 μF Capacitors: C_2, C_4, C_6, C_8
- 100 μF Capacitors: C_1
- 5 kΩ Trimmer Potentiometer: POT
- Arduino Uno Rev. 3: ARDUINO
- Arduino Stackable Header Kit
- Hamamatsu S1223 - Si PIN photodiode: PD
- IVC102 - Precision Switched Integrator Transimpedance Amplifier: IVC102
- LM1086 - 5.0V 1.5A Low Dropout Positive Regulator: LM1086
- LTC2440 - 24-Bit High Speed Differential $\Delta\Sigma$ ADC with Selectable Speed/Resolution: LTC2440
- Micro SD Card Breakout Board: SDCARD
- Raster Signal wire-board 1.25mm, 4 pin Socket: RASTER

A.2.2 Schematics

In this section we present the schematic for the 24 bit version of the acquisition circuit, pIDDO-24bit v1.6.

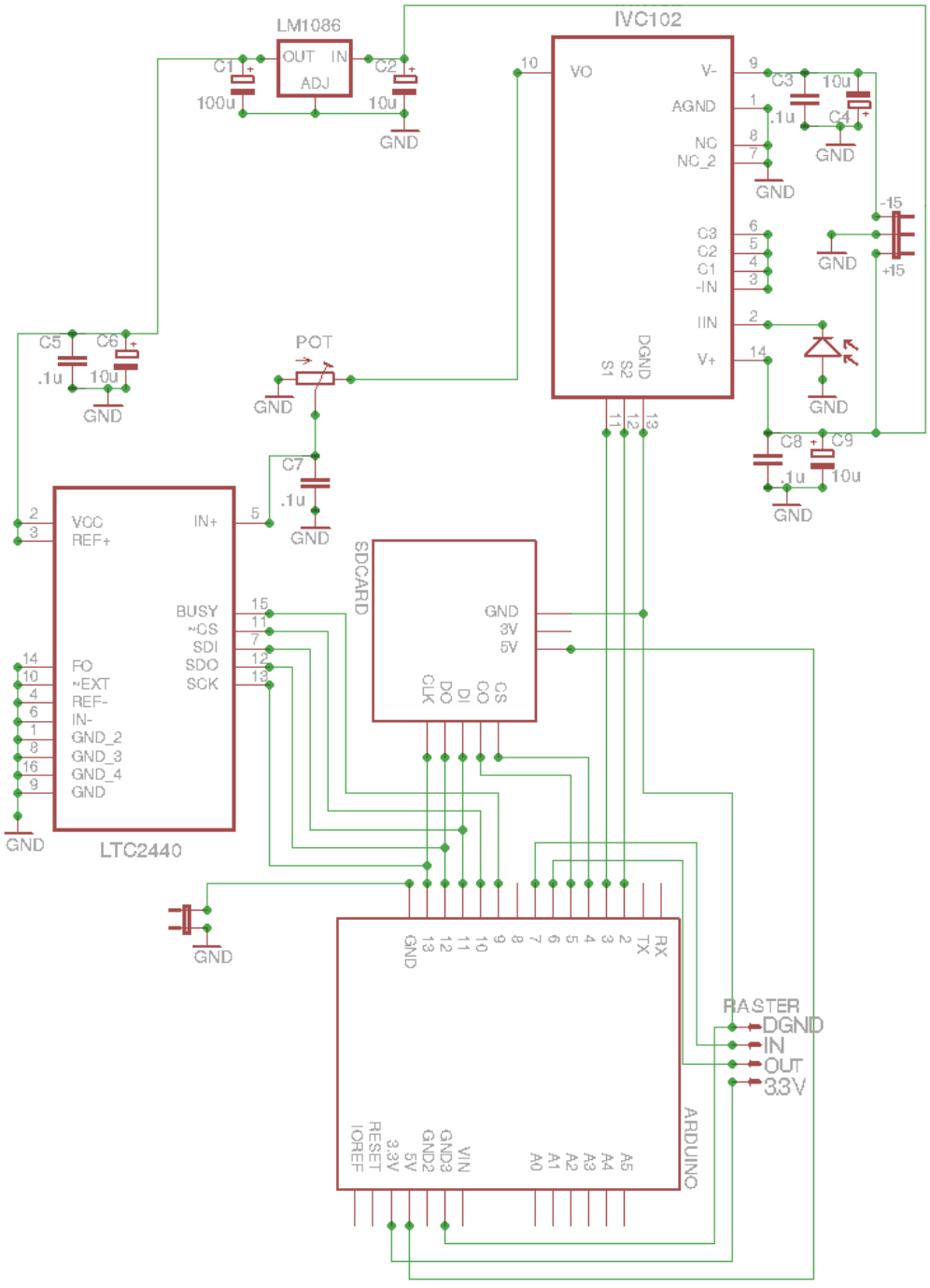


Figure A.4: *pIDDO-24bit v1.6 schematic.*

A.2.3 PCB

In this section we present the bottom and top layers of the PCB for the 24 bit version of the acquisition circuit, pIDDO-24bit v1.6.

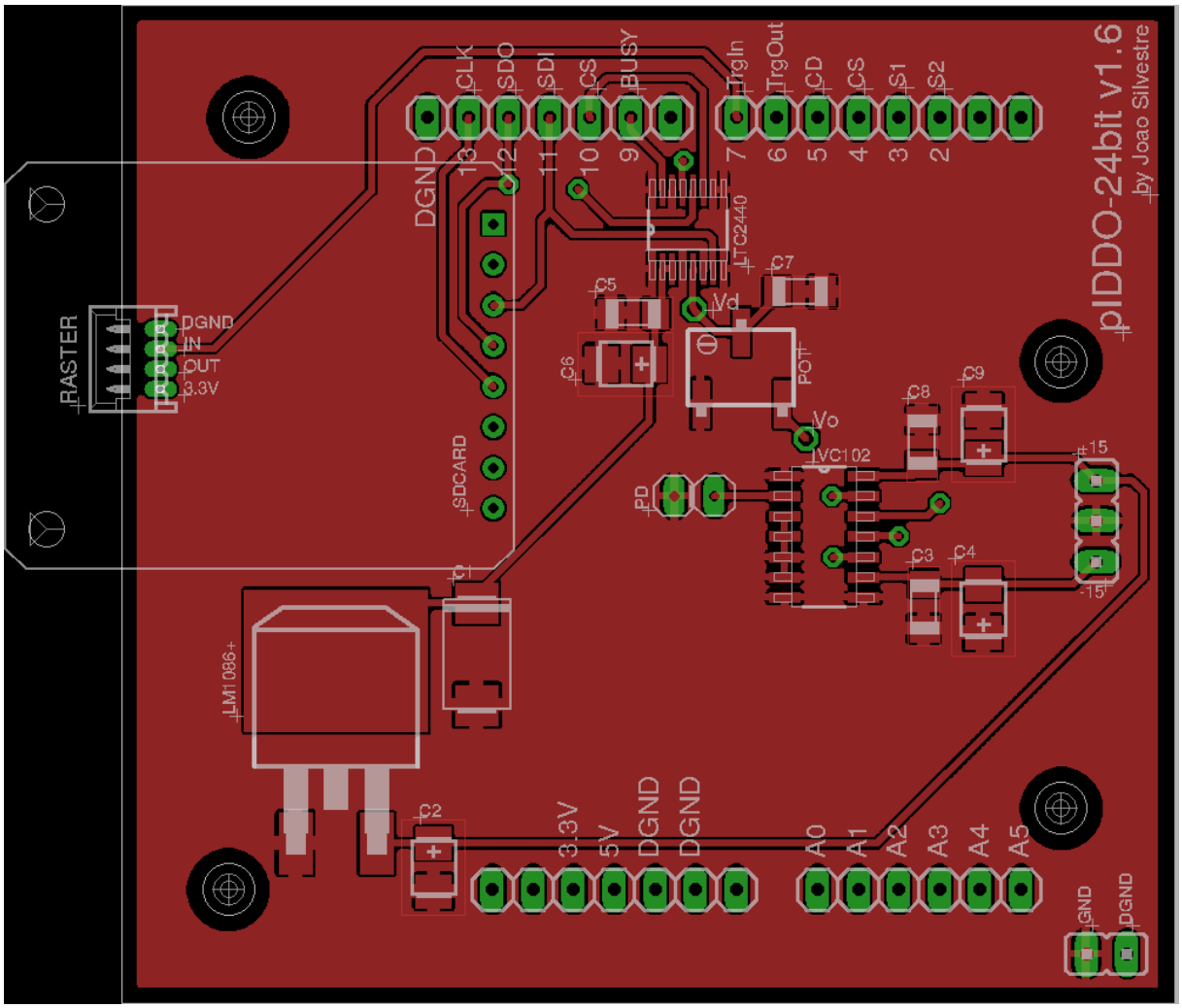


Figure A.5: *pIDDO-24bit v1.6 PCB top layer.*

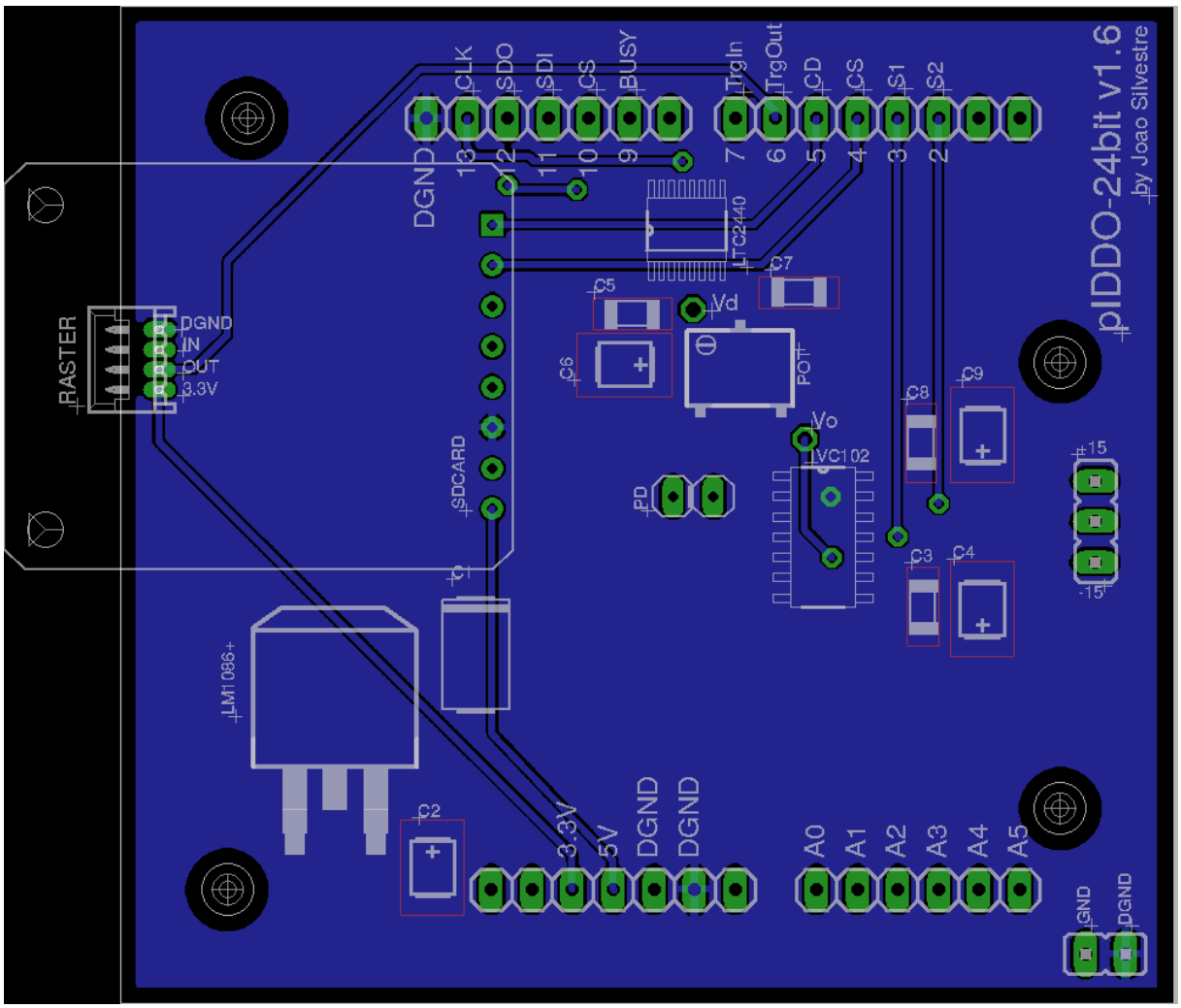


Figure A.6: *pIDDO-24bit v1.6 PCB bottom layer.*

A.3 24bit ADC Shield for Arduino Uno

In this appendix we present a bi-product of this dissertion, a 24 bit ADC shield for Arduino Uno. This shield enhances the performance of any Arduino Uno as this microprocessor's ADC pins can only resolve up to 10 bits.

A.3.1 Part List

In this section we present a list of parts, with their board labels, used in the production of the 24 bit ADC shield for Arduino Uno.

- 0.1 μF Capacitors: C_1, C_3
- 10 μF Capacitors: C_2
- Arduino Uno Rev. 3: ARDUINO
- Arduino Stackable Header Kit
- LTC2440 - 24-Bit High Speed Differential $\Delta\Sigma$ ADC with Selectable Speed/Resolution: LTC2440

A.3.2 Schematics

In this section we present the schematic for the 24 bit ADC shield for Arduino Uno.

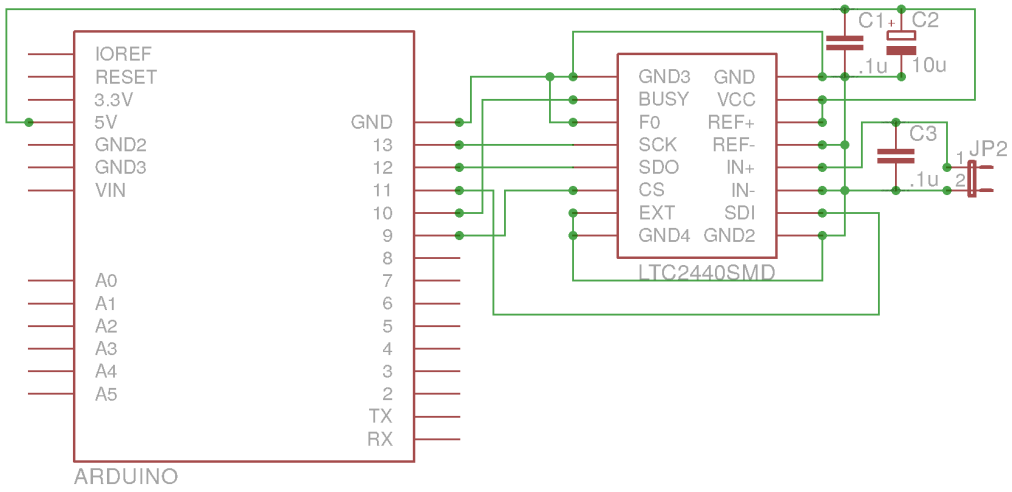


Figure A.7: *Arduino Uno 24bit ADC shield's schematic.*

A.3.3 PCB

In this section we present the bottom and top layers of the PCB for the 24 bit ADC shield for Arduino Uno.

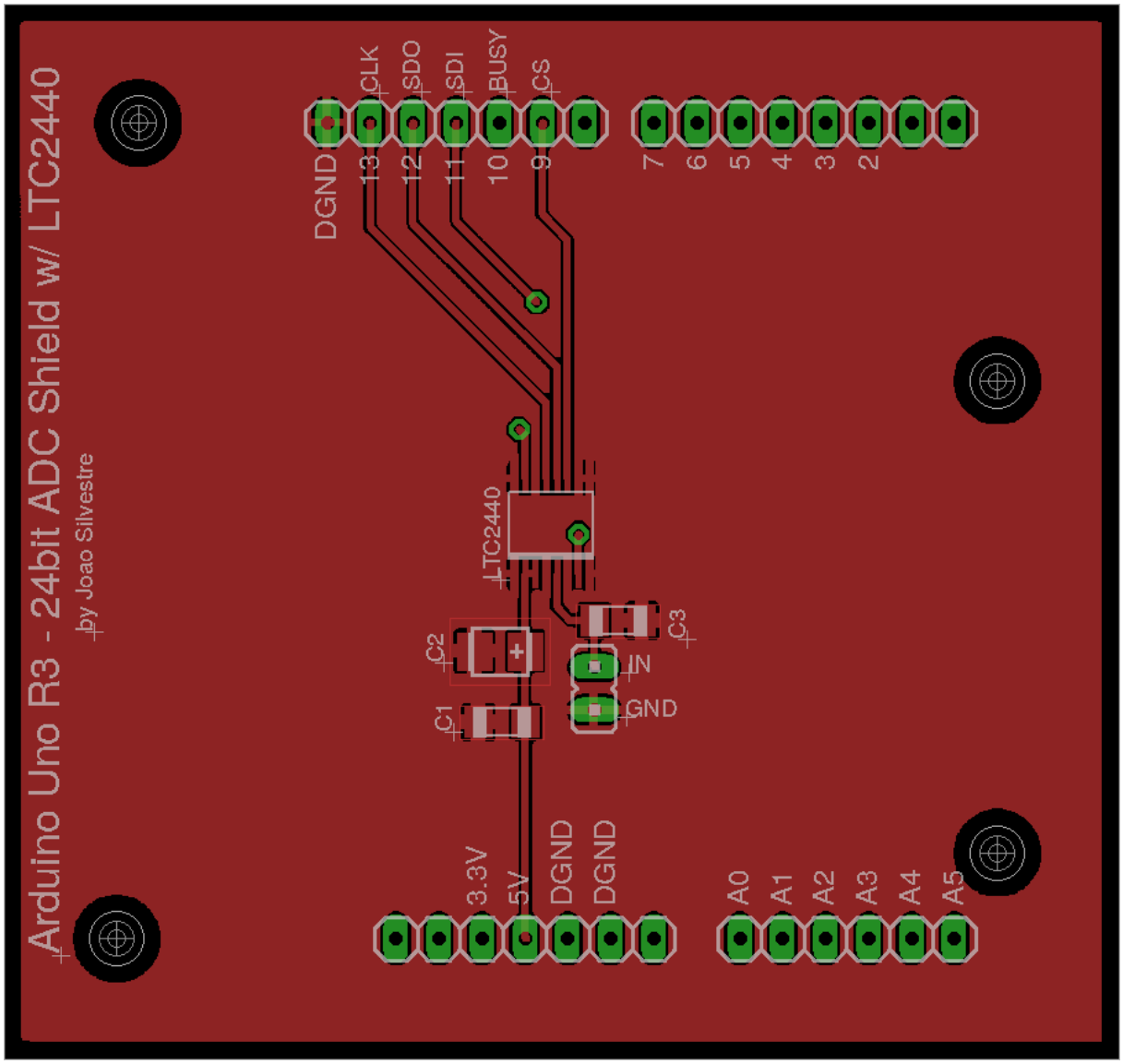


Figure A.8: *Arduino Uno 24bit ADC shield's PCB top layer.*

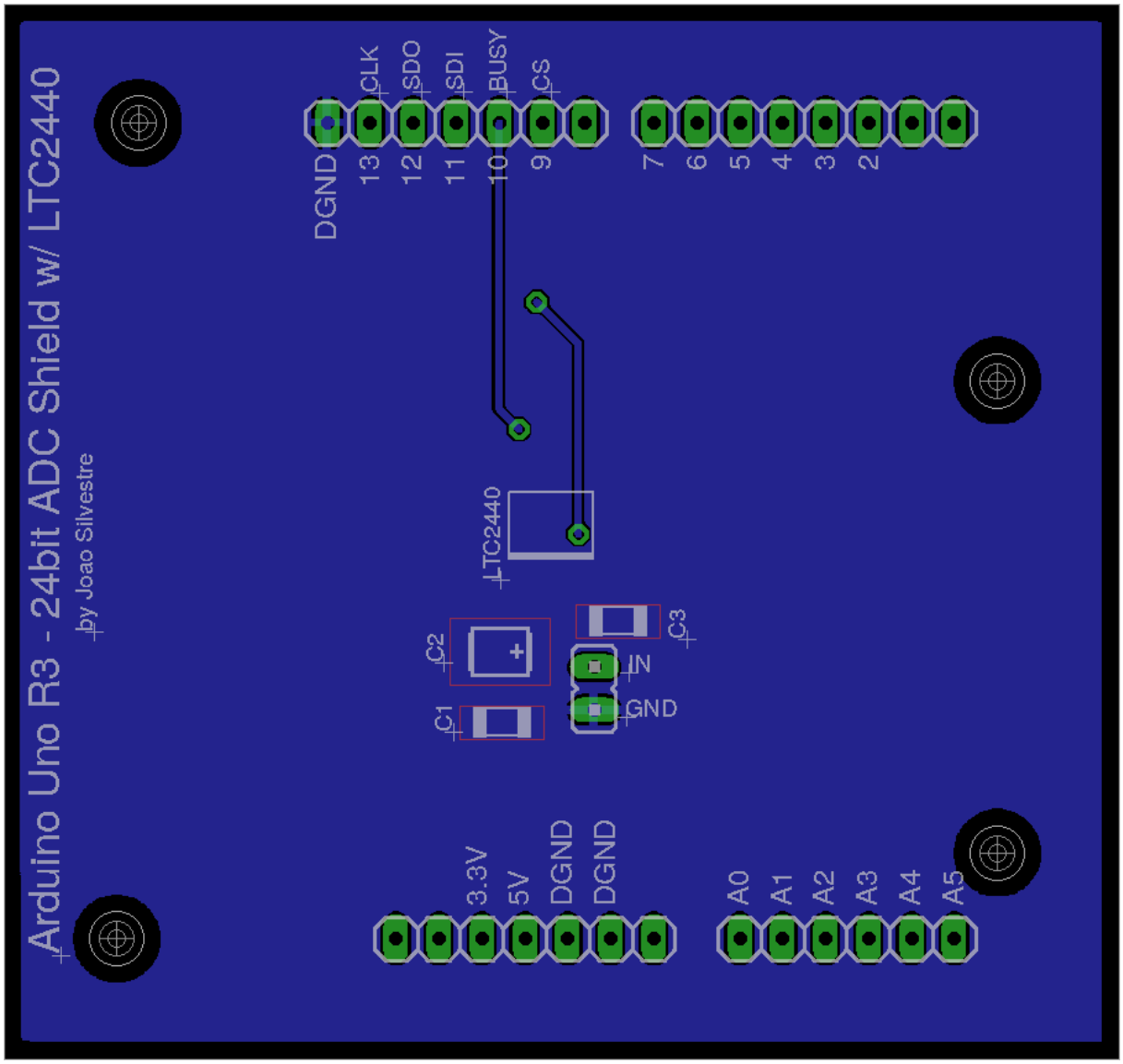


Figure A.9: *Arduino Uno 24bit ADC shield's PCB bottom layer.*

Appendix B

Instruments and Materials

In this appendix we list all materials, equipment, software and websites used and/or consulted, over the course of this dissertion, during the development of COSAC and its subsystems.

B.1 Materials

- 16× ADC LTC2440
- 1× Aluminum Sheet 600×350×25 mm
- 1× Aluminum Sheet 360×330×20 mm
- 1× Aluminum Sheet 330×320×15 mm
- 1× Aluminum Sheet 390×340×0.75 mm
- 4× Aluminum Sheet 320×320×0.75 mm
- 5× Arduino Uno R3
- 2× Arduino Stackable Header Kit - R3
- 1× BNC 50Ω Cable
- 3× Breadboard 30×10 pin
- 2× Breadboard 63×10 pin
- 4× Buffer LTC2051
- 6× Capacitor 0.01μF SMD
- 50× Capacitor 0.1μF SMD
- 6× Capacitor 1μF SMD
- 30× Capacitor 10μF SMD
- 5× Capacitor 100μF SMD
- 2× DLP LightCrafter Evaluation Module

- 6× Electric Wire 7.5 m, ∅0.75 mm
- 4× Fuse 2 A
- 1× Hamamatsu S1223 Photodiode
- 1× IEC C13 (female) 3 pin plug, 6A
- 1× IEC C14 (male) 3 pin socket, 4A, with switch and fuse box
- 6× Integrator IVC102
- 4× Inverter LF411
- 4× Inverter LF412
- 4× Isopropyl Alcohol 1L
- 1× Laser Pointer
- 1× LED 950 nm
- 1× Lens CSOPTLENS01 - 111-0210E
- 2× Lens CSOPTLENS01 - 111-0222E
- 3× MakerBeam Linear Slide Carriage
- 1× MakerBeam Linear Slide Rail
- 8× M2 Spacer 5 mm
- 20× M2.5 Bolt 10 mm
- 20× M2.5 Nut
- 50× M3 Bolt 10 mm
- 50× M3 Self Locking Nut
- 8× M3 Spacer 10 mm
- 16× M3 Square Bolt 12 mm
- 50× M3 Square Bolt 5 mm
- 70× M3 Nuts
- 4× OpenBeam 45 mm
- 1× OpenBeam 270 mm
- 4× OpenBeam 300 mm
- 12× OpenBeam Corner Cubes 15×15×15 mm
- 8× PCB Header 40pin 2 mm

- 5× PCB pIDDO v1.0
- 2× PCB pIDDO v1.1
- 2× PCB pIDDO v1.3
- 5× PCB pIDDO v1.4
- 3× Photopolymer Resin 1L
- 5× Raster Signal wire-board 1.25 mm, 4 pin Plug
- 5× Raster Signal wire-board 1.25 mm, 4 pin Socket
- 20× Raster Signal wire-board 1.25 mm, Terminal
- 4× Regulator 5 V LM1086
- 6× Resistor 10 Ω SMD
- 4× Resistor 59 Ω
- 2× Resistor 390 Ω
- 3× Resistor 1 kΩ SMD
- 3× Resistor 1.2 kΩ
- 2× Resistor 1.8 kΩ
- 10× Resistor 2.2 kΩ SMD
- 16× Resistor 5 kΩ SMD
- 4× Rubber feet 5 mm ∅14 mm
- 1× SD card Breakout Board, Adafruit
- 4× SIL 40Way Header
- 4× SIL 40Way Socket
- 4× SOIC-DIP 14 pin Breakout Board
- 1× Solder Resin 35 g
- 1× Solder Wire 0.5 mm, Sn99.3 Cu0.7 Alloy
- 8× SSOP-DIP 16 pin Breakout Board
- 3× Symmetrical 1 A Power Source Kit K8042
- 3× Thermoretractile Sleeve 1 m, ∅6.4 mm>3.2 mm
- 2× Toroid Transformer 230 V/24+24 V, 1.66+1.66 A, 80 VA
- 3× Trimmer Potentiometer 2 kΩ, 11 turn, SMD
- 3× Trimmer Potentiometer 5 kΩ, 2 turn
- 6× Trimmer Potentiometer 5 kΩ, 15 turn, SMD

B.2 Instruments

- 3D Printer, Form 1+
- Abrasive cut-off machine, Ryobi ECO-2335
- Angle grinder, Varo POW204
- Angle grinder station, Parkside
- Anti-static work station, ESD
- Asus N56VZ
- CNC (3-axes), Pronum, equipped with a Kress 1050 FME milling machine
- De-soldering pump, Pace ST-75
- Digital multimeter, KEYSIGHT 34461A
- Drilling press, Dewhurst & Partner LTD, Q&S QDM1250
- Drills (1.5 mm, 2 mm, 2.5 mm, 3 mm, 3.5 mm, 4 mm)
- Dremel 4000
- Lamp Velleman VTLAMP3WN
- Milling press (3-axes), Schaffner W12
- Mills (1 mm, 1.2 mm, 2 mm, 3 mm, 6 mm, 8 mm, 10 mm, 12 mm)
- Optical table Newport
- Oscilloscope Rigol DS6102
- Oscilloscope Tektronix TDS2014
- Photodiode PDA 100a
- Power source TENMA R 72-2535
- Soldering station Weller R WXD 2
- Steel ruler, 150 mm, 0.5 mm, Tajima 011M
- Ultrasonic Cleaner, VWR USC-T

B.3 Software

- Arduino 1.8.8
- CNC 4.01
- DLP LightCrafter GUI 5.0.0

- EAGLE 7.2.0 Light
- GIMP 2.8.22
- LibreOffice Calc 5.4.4.2
- PreForm
- R 3.4.3
- Solidworks Student Edition 2015
- Ubuntu 14.04 LTS
- VCarve Pro V6.5

B.4 Websites

- *http://dangerousprototypes.com/forum/*
- *https://www.eurocircuits.com/*
- *https://e2e.ti.com/*
- *http://www.ifastpcb.com/*
- *https://stackoverflow.com/*
- *https://trello.com/*
- *https://www.farnell.com/*
- *https://www.overleaf.com/*
- *https://www.ptrobotics.com/*

Appendix C

Control Programs

In this appendix we present the code for all programs developed and/or used over the course of this dissertation, additionally, for the most extensive, a flowchart is also displayed.

C.1 DMD-CS.cpp Flow Chart

In this section we present a flow chart of the processes and interactions of the signal coding control program, `DMD-CS.cpp`, with the DM365, the acquisition control program, `pIDDO-24bit.ino` and `pIDDO-10bit.ino`, and the user.

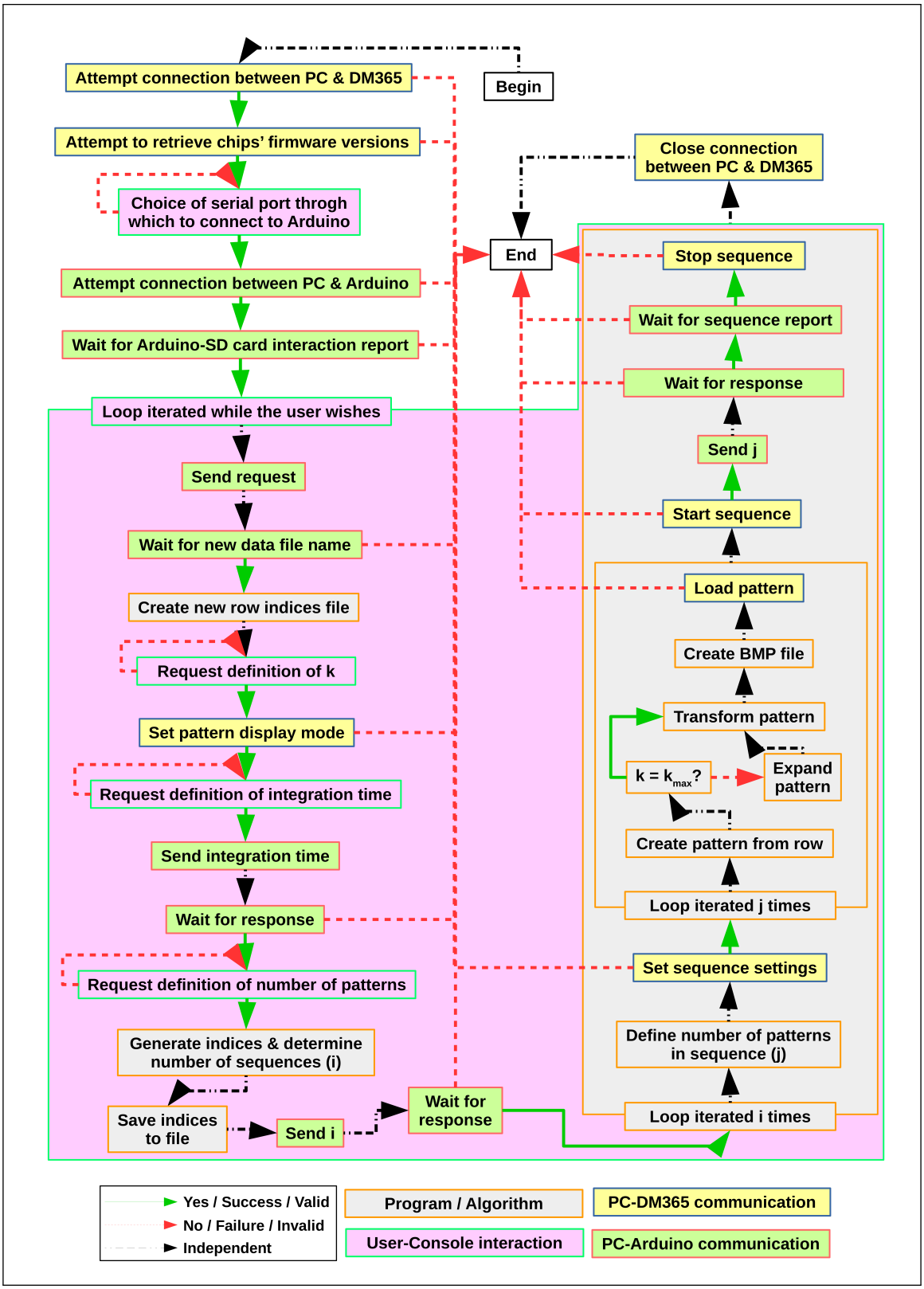


Figure C.1: *Simplified flowchart of tasks and interactions of the signal coding control program.*

C.2 DMD-CS.cpp

In this section we present the code of the program responsible for coordinating the signal coding subsystem.

```
/*
 * COSAC Control-PC Side. The purpose of this program is to handle pattern
 * generation, sequence set up, and communication with DM365.
 *
 * Copyright (C) 2018 Joao Rino Silvestre
 *
 * This program is free software; you can redistribute it and/or modify it under
 * the terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 2 of the License, or (at your option) any later
 * version.
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
 */

#include <cstdint>
#include <cstring>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <arpa/inet.h>
#include <math.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <time.h>
#include "rs232.h"
#include "BMP.h"
#include "PKT.h"
#include "SHELF.h"

main(){
  const int ver_Size=64;
  unsigned int pkt_length; //will hold the length of the packet
  int16 data_length;

  double timer;

  const int conHold = 1000000; //hold_time after connection is established (in \mus)
  const int seqDispHold = 3000000; //hold_time after all patterns have been displayed
      (in \mus)
  const int endPatHold = 2000000; //hold_time after the run is finished
```

```
int8 *pkt = new int8 [HEADER+MAX+CHECK]; //will hold the packets
string chip_name[3] = {"DM365: ","FPGA: ","MSP 430 SW: "};
int8 chip_cmd_code[3] = {0 /*dm365*/, 0x10 /*fpga*/, 0x20 /*msp430*/};
char (*ver_print)[ver_Size] = new char [3][ver_Size]; //will hold the versions of
    each chip of the DMD

/* Commands to be used */
int16 cmd_getVer=0x0100;
int16 cmd_CurDispMode=0x0101;
int16 cmd_StaticImg=0x0105;
int16 cmd_PatSeqSet=0x0400;
int16 cmd_ExtPatSeqSet=0x0480;
int16 cmd_PatDef=0x0401;
int16 cmd_ExtPatDef=0x0481;
int16 cmd_StartPatSeq=0x0402;
int16 cmd_NextPat=0x0403;

/* Socket Structure */
char ip[] = "192.168.1.100";
struct sockaddr_in sa;
sa.sin_family=AF_INET;
sa.sin_addr.s_addr=inet_addr(ip);
sa.sin_port = htons(0x5555);
int dm365 = socket(AF_INET,SOCK_STREAM,0);

/* Communication with Arduino */
int port_num;
string Query_0="To which port is the Arduino connected: ttyACM0 (0) or ttyACM1
    (1)?";
long int baud_rate=115200;
const char mode[]={'8','N','1',0};
const char trigger=' ';
string port_name;
unsigned char * Serial_buffer=new unsigned char[ver_Size];
Serial_buffer[0]='\0';
string sendstr;

/* Files */
ofstream indexes;

/*                              */
/* Attempt to Establish Connection */
/* ############################### */
if (connect(dm365,(sockaddr*) &sa,sizeof(sa))!=-1){
  cout<<"Connected."<<endl;
  cout<<" "<<endl;
}
else{
  cout<<"Connection failed."<<endl;
  delete [] pkt; delete [] Serial_buffer;
```

```
  return -1;
}

usleep(conHold);
/* ###################################### */
/* End of Attempt to Establish Connection */
/*                                        */


/*                                        */
/* Request DM365, FPGA & MSP430 Versions */
/* ###################################### */
for(int i=0; i<3; i++){
  data_length=1;
  Fill_PKT_Header(READ,cmd_getVer,SINGLE,data_length,pkt);
  pkt[HEADER]=chip_cmd_code[i];
  pkt[HEADER+data_length]=calcChecksum(HEADER+data_length, pkt);
  pkt_length = HEADER+data_length+CHECK;

  if(pkt_talk(dm365,pkt,pkt_length,data_length)==-1){
    cout<<"CMD FAILED."<<endl;
    delete [] pkt; delete [] Serial_buffer;
    return -1;
  }

  memcpy(ver_print[i],(char *)pkt+HEADER,ver_Size); //copies the contents of the
      received packet into the versions array
}
/* ###################### */
/* End of Version Request */
/*                        */

cout<<"COSAC Control-PC Side. The purpose of this program is to handle pattern
    generation,"<<endl;
cout<<"sequence set up, and communication with the DM365 processor present in
    the"<<endl;
cout<<"DLP LightCrafter Evaluation Module."<<endl<<endl;
cout<<"Copyright (C) 2018 Joao Rino Silvestre"<<endl<<endl;
cout<<"Welcome to the COSAC's LightCrafter Control Program!"<<endl<<endl<<endl;

/*              */
/* Print Versions */
/* ############## */
cout<<"LightCrafter software and firmware versions:"<<endl<<endl;

for(int i=0; i<3; i++)
  cout<<chip_name[i]<<" "<<ver_print[i]<<endl;
cout<<endl<<endl<<endl;
/* ##################### */
/* End of Print Versions */
/*                       */
```

```
/*                      */
/* Serial Port Definition */
/* ###################### */
if(Binary_Choice(Query_0)==0)
  port_name="ttyACM0";
else port_name="ttyACM1";

const char * portName=port_name.c_str();

port_num=RS232_GetPortnr(portName);
/* ############################## */
/* End of Serial Port Definition */
/*                            */

int PKT_Number, had_Order, rank, byteCount;
char bmp_name[32];
string Query_1="Do you wish to: Proceed (0) or Disconnect (1)?";
int bmp_payload_length=DMD_AREA/8;
int bmp_length=bmp_payload_length+OFFSET;
long int intTime;
int *powers;

/* Open Serial Port */
if(RS232_OpenComport(port_num,baud_rate,mode)==1){
  cout<<"Error opening serial port!"<<endl;
  delete [] pkt; delete [] Serial_buffer;
  return -1;
}

cout<<"Serial port succefully opened."<<endl;
cout<<"Communication with Arduino has been established."<<endl;

usleep(2000000);

/*                                  */
/* Start of Com-Session 1 with Arduino */
/* ################################### */

/* request and read first message of setup */
sendstr = "S";

RS232_cputs(port_num,sendstr.c_str());
RS232_flushTX(port_num);

while(RS232_PollComport(port_num,Serial_buffer,ver_Size)<=0)
  usleep(500);

if(Serial_buffer[0]!='Y'){
  cout<<"Error: Couldn't initialize the SD Card."<<endl;
  delete [] pkt; delete [] Serial_buffer;
```

```
  return -1;
}
cout<<"SD Card initialization was successful."<<endl<<endl;
/* ################################### */
/* End of Com-Session 1 with Arduino */
/*                                   */

/*                  */
/* Start of Main Loop */
/* ################## */

while(Binary_Choice(Query_1)==0){

  int *line_Numbers;
  int8 *had_Line;
  int8 *transf_had_Line=new int8[DMD_AREA];

  for(int ww=0; ww<DMD_AREA; ww++)
    transf_had_Line[ww]=0;

  had_Order = 18;
  intTime = 0;
  timer = time(NULL);
  string indexFilename = "I";
  string dataFilename = "\0";
  string sendstr1;

  /*                                   */
  /* Start of Com-Session 2 with Arduino */
  /* ################################### */

  /* request and read name of data file to be created */
  sendstr = "F";
  RS232_cputs(port_num,sendstr.c_str());
  RS232_flushTX(port_num);

  usleep(100000);
  while(RS232_PollComport(port_num,Serial_buffer,ver_Size)<=0){
    usleep(100000);
  }

  if(Serial_buffer[0]!='?'){
    cout<<"Error: unexpected response."<<endl;
    delete [] pkt; delete [] Serial_buffer;
    delete [] line_Numbers; delete [] had_Line;
    delete [] transf_had_Line;
    return -1;
  }

  for(int i=1; Serial_buffer[i]!='!'; i++)
    dataFilename += Serial_buffer[i];
```

```
cout<<"The data will be saved in the SD Card to: "<<dataFilename<<endl;
/* ################################# */
/* End of Com-Session 2 with Arduino */
/*                                   */

indexFilename += dataFilename;

cout<<"The indexes will be saved in the execution folder to:
    "<<indexFilename<<endl<<endl;

// The DMD pixel array is a diamond grid 608x684. In order to represent a rotated
    square in it, that square max side is 256 (currently), 256^2==2^16
while(had_Order>MAP_ORDER_MAX || (had_Order%2)!=0 || had_Order<4){ // safeguard
    loop
  cout<<"Choose the order, k, for an Hadmard matrix of rank 2^k (k must be even,
      larger than 4 and smaller than 18)."<<endl;
  cin>>had_Order;
}
cout<<endl;

indexes.open(indexFilename);
indexes<<had_Order<<endl;

rank=pow(2,had_Order);
had_Line = new int8 [rank];
powers = new int[had_Order-1];
powers[0] = 1;

for(int p=1; p<had_Order;p++)
  powers[p]=powers[p-1]*2;

/*                                   */
/* Set Display Mode to Pattern Sequence */
/* ################################## */
data_length=1;
pkt_length=HEADER+data_length+CHECK;
Fill_PKT_Header(WRITE,cmd_CurDispMode,SINGLE,data_length,pkt);
pkt[HEADER]=PATTERN_SEQ;
pkt[HEADER+data_length]=calcChecksum(HEADER+data_length,pkt);

if(pkt_talk(dm365,pkt,pkt_length,data_length)==-1){
  cout<<"CMD FAILED."<<endl;
  delete [] pkt; delete [] Serial_buffer; delete [] powers;
  delete [] had_Line; delete [] transf_had_Line;
  return -1;
}
/* ######################################### */
/* End of Set Display Mode to Pattern Sequence */
/*                                           */
```

```
int pat_Number = 0; //number of patterns to use
int seq_Max = 96; // max number of patterns per sequence
int seq_Lim; //number of patterns in the sequence
int8 *exp_had_Line = new int8[MAP_MAX];
PKT_Number = ceil((double)bmp_length/(MAX-1)); // one byte is saved for pattern
    number in sequence

while(intTime<=0){
  cout<<"Type in the integration time interval, in us: ";
  cin>>intTime;
}
cout<<endl;

/*                                  */
/* Start of Com-Session 3 with Arduino */
/* ################################## */

/* send number of patterns and wait for response */
sendstr1 = '?'+to_string(intTime)+'!';
RS232_cputs(port_num,sendstr1.c_str());
RS232_flushTX(port_num);

while(RS232_PollComport(port_num,Serial_buffer,ver_Size)<=0)
  usleep(500);

if(Serial_buffer[0]!='Y'){
  cout<<"Error: unexpected response."<<endl;
  delete [] pkt; delete [] Serial_buffer;
  delete [] line_Numbers; delete [] had_Line;
  delete [] transf_had_Line; delete [] exp_had_Line;
  return -1;
}
/* ################################## */
/* End of Com-Session 3 with Arduino */
/*                                  */

while(pat_Number<=0 || pat_Number>=rank){
  cout<<"Type in the number of patterns to use (must be smaller than, or equal to,
      "<<rank<<"):"<<endl;
  cin>>pat_Number;
}
cout<<endl;

line_Numbers=new int[pat_Number];

if(pat_Number==rank)
  for(int i=0; i<pat_Number; i++)
    line_Numbers[i]=i;
else
  Random_Lines(timer,pat_Number,line_Numbers,rank);
```

```
for(int i=0; i<pat_Number; i++) // saving indexes used to file
  indexes<<line_Numbers[i]<<endl;;
indexes.close();

cout<<"The indexes have been generated and saved to file."<<endl<<endl;

int seq_Number = ceil((double)pat_Number/seq_Max);
int seq_Last = seq_Number-1;
int seq_Offset;

/*                                */
/* Start of Com-Session 4 with Arduino */
/* ################################## */

/* send number of patterns and wait for response */
sendstr1 = '?'+to_string(pat_Number)+'!';
RS232_cputs(port_num,sendstr1.c_str());
RS232_flushTX(port_num);

while(RS232_PollComport(port_num,Serial_buffer,ver_Size)<=0)
  usleep(500);

if(Serial_buffer[0]!='Y'){
  cout<<"Error: unexpected response."<<endl;
  delete [] pkt; delete [] Serial_buffer;
  delete [] line_Numbers; delete [] had_Line;
  delete [] transf_had_Line; delete [] exp_had_Line;
  return -1;
}
/* ################################## */
/* End of Com-Session 4 with Arduino */
/*                                */

/*                    */
/* Sequence Display Loop */
/* ##################### */
for(int q=0; q<seq_Number; q++){

  if(q==0 && seq_Number!=1){
    seq_Lim = seq_Max;

    /*                        */
    /* Pattern Sequence Setting */
    /* ######################## */
    data_length=17;
    pkt_length=HEADER+data_length+CHECK;
    Fill_PKT_Header(WRITE,cmd_PatSeqSet,SINGLE,data_length,pkt);
    PKT_Seq_Set(seq_Lim,EXTERNAL_POS,RED,pkt);
    pkt[HEADER+data_length]=calcChecksum(HEADER+data_length,pkt);

    if(pkt_say(dm365,pkt,pkt_length)==-1){
```

```
    cout<<"CMD FAILED."<<endl;
    delete [] pkt; delete [] Serial_buffer; delete [] powers;
    delete [] line_Numbers; delete [] had_Line;
    delete [] transf_had_Line; delete [] exp_had_Line;
    return -1;
  }
  /* ################################ */
  /* End of Pattern Sequence Setting */
  /*                                  */
}

if(q==seq_Last){
  seq_Lim = pat_Number%seq_Max;

  /*                         */
  /* Pattern Sequence Setting */
  /* ######################## */
  data_length=17;
  pkt_length=HEADER+data_length+CHECK;
  Fill_PKT_Header(WRITE,cmd_PatSeqSet,SINGLE,data_length,pkt);
  PKT_Seq_Set(seq_Lim,EXTERNAL_POS,RED,pkt);
  pkt[HEADER+data_length]=calcChecksum(HEADER+data_length,pkt);

  if(pkt_say(dm365,pkt,pkt_length)==-1){
    cout<<"CMD FAILED."<<endl;
    delete [] pkt; delete [] Serial_buffer; delete [] powers;
    delete [] line_Numbers; delete [] had_Line;
    delete [] transf_had_Line; delete [] exp_had_Line;
    return -1;
  }
  /* ################################ */
  /* End of Pattern Sequence Setting */
  /*                                  */
}

seq_Offset = q*seq_Max;

/*                   */
/* Pattern Definition */
/* ################## */
for(int r=0; r<seq_Lim; r++){
  Create_Hadamard_Lines(line_Numbers[seq_Offset+r],had_Order,had_Line,powers);

  if(rank<MAP_MAX){
    Expand_Hadamard_Map(had_Line,exp_had_Line, rank);
    Transform_Map(transf_had_Line,exp_had_Line);
  }
  else
    Transform_Map(transf_had_Line,had_Line);

  if(PKT_Number==1){
```

```
  pkt[HEADER]=(int8)r;
  data_length=bmp_length+1;
  pkt_length=HEADER+data_length+CHECK;
  Fill_PKT_Header(WRITE,cmd_PatDef,SINGLE,data_length,pkt);

  if(Fill_BMP_Array(pkt+HEADER+1,transf_had_Line,bmp_length,bmp_payload_length)==-1){
    cout<<"Failed to convert hadamard line to BMP format."<<endl;
    delete [] pkt; delete [] Serial_buffer; delete [] powers;
    delete [] line_Numbers; delete [] had_Line;
    delete [] transf_had_Line; delete [] exp_had_Line;
    return -1;
  }

  pkt[HEADER+data_length]=calcChecksum(HEADER+data_length,pkt);

  if(pkt_say(dm365,pkt,pkt_length)==-1){
    cout<<"CMD FAILED."<<endl;
    delete [] pkt; delete [] Serial_buffer; delete [] powers;
    delete [] line_Numbers; delete [] had_Line;
    delete [] transf_had_Line; delete [] exp_had_Line;
    return -1;
  }
}
else{
  if(PKT_Number>1){ // only needed for k=20
    int pkt_flag;
    int8 * pkt_buf=new int8 [bmp_length];
    pkt_buf[0]=(int8)r;

    if(Fill_BMP_Array(pkt_buf+1,transf_had_Line,bmp_length,bmp_payload_length)==-1){
      cout<<"Failed to convert hadamard line to BMP format."<<endl;
      delete [] pkt; delete [] Serial_buffer; delete [] powers;
      delete [] line_Numbers; delete [] had_Line;
      delete [] transf_had_Line; delete [] exp_had_Line;
      return -1;
    }

    for(int p=0; p<PKT_Number; p++){
      if(p==0){
        pkt_flag=FIRST;
        data_length=MAX;
      }
      if(p>0 && p<PKT_Number-1){
        pkt_flag=MID;
        data_length=MAX;
      }
      if(p==PKT_Number-1){
        pkt_flag=LAST;
        data_length=(bmp_length % (MAX-1))+1;
      }
```

```
        pkt_length=HEADER+data_length+CHECK;
        Fill_PKT_Header(WRITE,cmd_PatDef,pkt_flag,data_length,pkt);
        memcpy((char *)pkt+HEADER,(char *)pkt_buf+p*MAX,data_length);
        pkt[HEADER+data_length]=calcChecksum(HEADER+data_length,pkt);

        if(pkt_say(dm365,pkt,pkt_length)==-1){
          cout<<"CMD FAILED."<<endl;
          delete [] pkt; delete [] Serial_buffer; delete [] powers;
          delete [] line_Numbers; delete [] had_Line;
          delete [] transf_had_Line; delete [] exp_had_Line;
          return -1;
        }
      }
    }
    else{
      cout<<"Error: Invalid number of packets"<<endl;
      delete [] pkt; delete [] Serial_buffer; delete [] powers;
      delete [] line_Numbers; delete [] had_Line;
      delete [] transf_had_Line; delete [] exp_had_Line;
      return -1;
    }
  }
}
/* ########################## */
/* End of Pattern Definition */
/*                          */

/*                */
/* Pattern Sequence */
/* ################ */

/* START */
data_length=1;
pkt_length=HEADER+data_length+CHECK;
Fill_PKT_Header(WRITE,cmd_StartPatSeq,SINGLE,data_length,pkt);
pkt[HEADER]=START;
pkt[HEADER+data_length]=calcChecksum(HEADER+data_length,pkt);

if(pkt_say(dm365,pkt,pkt_length)==-1){
  cout<<"CMD FAILED."<<endl;
  delete [] pkt; delete [] Serial_buffer; delete [] powers;
  delete [] line_Numbers; delete [] had_Line;
  delete [] transf_had_Line; delete [] exp_had_Line;
  return -1;
}

usleep(350000); //this sleep is required as to prepare the dmd for triggers

/*                                  */
/* Start of Com-Session 5 with Arduino */
/* ################################### */
```

```
sendstr1 = '?'+to_string(seq_Lim)+'!';
RS232_cputs(port_num,sendstr1.c_str());
RS232_flushTX(port_num);

usleep(10000);
while(RS232_PollComport(port_num,Serial_buffer,ver_Size)<=0){
  usleep(500);
}

if(Serial_buffer[0]!='Y'){
  cout<<"Error: unexpected response."<<endl;
  delete [] pkt; delete [] Serial_buffer;
  delete [] line_Numbers; delete [] had_Line;
  delete [] transf_had_Line; delete [] exp_had_Line;
  return -1;
}

cout<<"Starting sequence "<<q+1<<"."<<endl;

while(RS232_PollComport(port_num,Serial_buffer,ver_Size)<=0)
  usleep(1000);

if(Serial_buffer[0]!='T'){
  cout<<"Error: unexpected response."<<endl;
  delete [] pkt; delete [] Serial_buffer;
  delete [] line_Numbers; delete [] had_Line;
  delete [] transf_had_Line; delete [] exp_had_Line;
  return -1;
}
/* ################################## */
/* End of Com-Session 5 with Arduino */
/*                                   */

/* STOP */
data_length=1;
pkt_length=HEADER+data_length+CHECK;
Fill_PKT_Header(WRITE,cmd_StartPatSeq,SINGLE,data_length,pkt);
pkt[HEADER]=STOP;
pkt[HEADER+data_length]=calcChecksum(HEADER+data_length,pkt);

if(pkt_say(dm365,pkt,pkt_length)==-1){
  cout<<"CMD FAILED."<<endl;
  delete [] pkt; delete [] Serial_buffer; delete [] powers;
  delete [] line_Numbers; delete [] had_Line;
  delete [] transf_had_Line; delete [] exp_had_Line;
  return -1;
}
/* ######################## */
/* End of Pattern Sequence */
/*                         */
```

```
      cout<<q*96+seq_Lim<<" patterns sent..."<<endl;
    }

    cout<<"All patterns have been displayed."<<endl<<endl;

    delete [] had_Line; delete [] transf_had_Line; delete [] line_Numbers;
    delete [] powers; delete [] exp_had_Line;
  }
  /* ################# */
  /* End of Main Loop */
  /*                */

  if(!close(dm365))
    cout<<endl<<"Disconnected."<<endl;

  RS232_CloseComport(port_num);

  delete [] pkt; delete [] Serial_buffer;
}
```

C.3 PKT.h

In this section we present a library of functions written to facilitate communication between the signal coding crontrol program and the DM365.

```
/*****************************************************************************
*   Copyright (C) 2013 Texas Instruments Incorporated - http://www.ti.com/ *
*****************************************************************************
*  Redistribution and use in source and binary forms, with or without *
*  modification, are permitted provided that the following conditions are *
*  met:                                                                   *
*                                                                         *
*  Redistributions of source code must retain the above copyright notice, *
*  this list of conditions and the following disclaimer.                  *
*                                                                         *
*  Redistributions in binary form must reproduce the above copyright notice, *
*  this list of conditions and the following disclaimer in the documentation *
*  and/or other materials provided with the distribution.                 *
*                                                                         *
*  Neither the name of Texas Instruments Incorporated nor the names of its *
*  contributors may be used to endorse or promote products derived from this *
*  software without specific prior written permission.                    *
*                                                                         *
*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS *
*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED *
*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A *
*  PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER *
*  OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, *
*  EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, *
*  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR *
*  PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANYTHEORY OF *
*  LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING *
*  NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS *
*  SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.       *
*****************************************************************************/

#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <unistd.h>
#include <string.h>

typedef unsigned char int8;
typedef unsigned short int16;

enum Type{BUSY, ERROR, WRITE, WRITE_RESP, READ, READ_RESP};
enum Flag{SINGLE, FIRST, MID, LAST};
enum PKT_Size{HEADER=6, MAX=0xFFFF, CHECK=1};
enum CurDispMode{STATIC,INTERNAL_PAT,HDMI,RESERVED,PATTERN_SEQ};
enum
    Trigger{COMMAND,AUTO,EXTERNAL_POS,EXTERNAL_NEG,CAMERA_POS,CAMERA_NEG,EXTERNAL_EXPOSURE};
```

```
enum Led{RED,GREEN,BLUE};
enum Control_Seq{STOP,START};

using namespace std;

/* Calculates sum of N components of an array (from 0 to N-1) */
int8 calcChecksum(unsigned int N, int8* packet){
  unsigned int sum = 0;

  for(unsigned int j = 0; j < N; j++)
    sum += packet[j];

  return (int8)(sum & 0xFF);
}


/* Sets the header for a PKT */
void Fill_PKT_Header(int8 type, int16 command, int8 flag, int16 Data_Length, int8*
    packet){
  packet[0]=type;                                                    // command type
  packet[1]=(command >> 8) & 0xFF; packet[2]=(command) & 0xFF; // command length
  packet[3]=flag;                                                    // flag about the
      ordinality of this packet
  packet[4]=Data_Length & 0xFF; packet[5]=(Data_Length >> 8) & 0xFF; // data length
}


/* Sets the body for PKT to carry pattern sequence settings */
void PKT_Seq_Set(int8 numberPat, int8 trigger, int8 led, int8* packet){
    packet[HEADER]=1;                          // bit depth: 1-8
    packet[HEADER+1]=numberPat;       // number of patterns in seq: 1-96
    packet[HEADER+2]=0;                        // use inverted patterns?
    packet[HEADER+3]=trigger;      // must change this to use trigger by arduino
    packet[HEADER+4]=0;                        // input trigger delay - LSB to HSB
    packet[HEADER+5]=0;                        //
    packet[HEADER+6]=0;                        //
    packet[HEADER+7]=0;                        //
    packet[HEADER+8]=0;                        // trigger period - LSB to HSB
    packet[HEADER+9]=0;                        //
    packet[HEADER+10]=0;                       //
    packet[HEADER+11]=0;                       //
    packet[HEADER+12]=0;                       // exposure time - LSB to HSB
    packet[HEADER+13]=0;                       //
    packet[HEADER+14]=0;                       //
    packet[HEADER+15]=0;                       //
    packet[HEADER+16]=led;             // led to be turned on
}


/* Sends a PKT into the DLP */
int pkt_say(int socket, int8* packet, unsigned int pkt_length){
```

```
//clock_t now=clock();
  unsigned int amount_written=write(socket,packet,pkt_length);

  if(amount_written!=pkt_length){
    cout<<"CMD reading failed."<<endl;
    return -1;}
//now=clock()-now;
//cout<<"time to say: "<<((float)now)/CLOCKS_PER_SEC<<endl;
  return 0;
}


/* Sends a PKT into the DLP and expects a response PKT from it */
int pkt_talk(int socket, int8* packet,unsigned int pkt_length, int16 data_length){

  unsigned int amount_written = write(socket,packet,pkt_length);
  int8 response_pkt_type=0;

  if(amount_written != pkt_length){
    cout<<"PKT writing failed."<<endl;
    return -1;}

  if(packet[0]==WRITE || packet[0]==READ)
    response_pkt_type=packet[0]+1;

  /* Retrieves DLP Response Header Info */
  if(read(socket,packet,HEADER) != HEADER){
    cout<<"CMD reading failed."<<endl;
    return -1;}

  /* Checks that the response PKT is of the expected type */
  if(packet[0] != response_pkt_type){
    cout<<"CMD failed."<<endl;
    return -1;}

  /* New Data Length */
  data_length = (int8)packet[4] | packet[5] << 8;

  /* Retrieves DLP Response Data */
  if(read(socket,packet+HEADER,data_length+1) != data_length+1){
    cout<<"Payload reading failed."<<endl;
    return -1;}

  /* CHECKSUM */
  if(packet[data_length+HEADER] != calcChecksum(data_length+HEADER,packet)){
    cout<<"Checksum Error"<<endl;
    return -1;}

  return 0;
}
```

C.4 BMP.h

In this section we present a library of functions written to perform manipulations and encoding of Hadamard rows to control the DMD grid.

```
/*
 * BMP.h. The present functions handle pattern generation and manipulation, and
 * parsing into BMP file format, accounting for how that BMP file will be
 * interpreted by the DM365.
 *
 * Copyright (C) 2017 Joao Rino Silvestre
 *
 * This program is free software; you can redistribute it and/or modify it under
 * the terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 2 of the License, or (at your option) any later
 * version.
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
 */

#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <string.h>
#include <time.h>
typedef unsigned char int8;
typedef unsigned short int16;

enum Defaul_Measures{DMD_HEIGHT=684,DMD_WIDTH=608,DEF_MAP_SIDE=256,MAP_MAX=256*256,
MAP_ORDER_MAX=16,DMD_AREA=684*608};
enum BMP_Measures{BMP_HEADER=14,DIB_HEADER=40,COLOR=8,OFFSET=62};

using namespace std;

/* Generates an array filled with 0s except for index position */
void Create_Raster_Lines(int index, int8* line, int rank){

  int lim=rank*rank;

  for(int i=0; i<lim; i++)
    line[i]=0;

  line[index]=1;
}
```

```
/* Generates line l (0 <= l <= 2^k -1) of an hadamard matrix of order k (rank=2^k)
   This matrix would be built according to sylvester construction */
void Create_Hadamard_Lines(int index, int k, int8* line, int* pots){

  line[0]=1;

  for(int n=0; n<k; n++){
    if((index/pots[n])%2==0)
      for(int i=0; i<pots[n]; i++)
        line[i+pots[n]]=line[i];
    else
      for(int i=0; i<pots[n]; i++)
        line[i+pots[n]]=(int8)(line[i]==0); // 1->1, -1->0
  }
}

/* Takes a squared pattern such that it's sides are divisors of max side size,
   and expands said pattern into a max_sidexmax_side pattern */
void Expand_Hadamard_Map(int8* OrigLine, int8* DefLine, int OrigSize){

  int sqos=sqrt(OrigSize);
  int ratio=DEF_MAP_SIDE/sqos;

  for(int i=0, l=0, m=0; i<OrigSize; i++){

    for(int j=0 ; j<ratio; j++)
      for(int k=0 ; k<ratio; k++)
        DefLine[(m+j)*DEF_MAP_SIDE+l+k]=OrigLine[i];

    if((i+1)%sqos==0){
      m+=ratio;
      l=0;
    }
    else l+=ratio;
  }
}


/* Centers 256x256 map line in 608x684 grid, then rotates the map 45 degrees */
/* HadLine is 256x256 */
/* Line is 608x684 */
void Transform_Map(int8* Line, int8* HadLine){
  int half_DMS=DEF_MAP_SIDE/2;
  int double_DMS=DEF_MAP_SIDE*2;
  int inc_DMS=DEF_MAP_SIDE+1;
  int new_nul=DMD_AREA/2+(DMD_WIDTH-DEF_MAP_SIDE)/2;
  int new_one=new_nul-DMD_WIDTH+1;              // defined to avoid checks as the
      diagonal walk is intermitent
  int new_n2r=new_nul+DMD_WIDTH+1;              //
```

```
  int new_o2r=new_nul+1;                          //
  int jump_h=2*DMD_WIDTH-1;
  int jump_v=2*DMD_WIDTH+1;
  int array_jump;
  int array_jump_v;
  int dmd_jump;
  int dmd_jump_v;

  for(int b=0; b<half_DMS; b++){
    array_jump_v=jump_v*b;
    dmd_jump_v=double_DMS*b;

    for(int c=0; c<half_DMS; c++){
      array_jump=array_jump_v-jump_h*c;
      dmd_jump=dmd_jump_v+2*c;

      Line[new_nul+array_jump]=HadLine[dmd_jump];
      Line[new_one+array_jump]=HadLine[1+dmd_jump];
      Line[new_n2r+array_jump]=HadLine[DEF_MAP_SIDE+dmd_jump];
      Line[new_o2r+array_jump]=HadLine[inc_DMS+dmd_jump];
    }
  }
}


/* Creates an array structured as a bmp */
int Fill_BMP_Array(int8* BMP_array, int8* pixel_map_array, int BMP_length, int
    BMP_payload_length){
  int b;

  if(BMP_payload_length%8!=0){
    cout<<"Error: your bmp array should include padding"<<endl;
    return -1;}

  /* Filling BMP HEADER */
  BMP_array[0]=0x42; BMP_array[1]=0x4d;
                                                        // BM
  BMP_array[2]=BMP_length & 0xff;             BMP_array[3]=(BMP_length >> 8) & 0xff;
           // BMP Size
  BMP_array[4]=(BMP_length >> 16) & 0xff;     BMP_array[5]=(BMP_length >> 24) & 0xff;
          //
  BMP_array[6]=0x00; BMP_array[7]=0x00;       BMP_array[8]=0x00; BMP_array[9]=0x00;
            // whatever
  BMP_array[10]=0x3e; BMP_array[11]=0x00;     BMP_array[12]=0x00; BMP_array[13]=0x00;
          // DATA offset
  /*Filling DIB HEADER */
  BMP_array[14]=0x28; BMP_array[15]=0x00;     BMP_array[16]=0x00; BMP_array[17]=0x00;
          // DIB HEADER SIZE
  BMP_array[18]=DMD_WIDTH & 0xff;             BMP_array[19]=(DMD_WIDTH >> 8) & 0xff;
           // Width
  BMP_array[20]=(DMD_WIDTH >> 16) & 0xff;     BMP_array[21]=(DMD_WIDTH >> 24) & 0xff;
```

```
        //
  BMP_array[22]=DMD_HEIGHT & 0xff;            BMP_array[23]=(DMD_HEIGHT >> 8) & 0xff;
        // Height
  BMP_array[24]=(DMD_HEIGHT >> 16) & 0xff;    BMP_array[25]=(DMD_HEIGHT >> 24) &
      0xff;    //
  BMP_array[26]=0x01; BMP_array[27]=0x00;
                                                      // Number of color planes
  BMP_array[28]=0x01; BMP_array[29]=0x00;
                                                      // Number of bits per pixel
  BMP_array[30]=0x00; BMP_array[31]=0x00;     BMP_array[32]=0x00; BMP_array[33]=0x00;
        // Compression

  BMP_array[34]=BMP_payload_length & 0xff;    BMP_array[35]=(BMP_payload_length >> 8)
      & 0xff; // Data Size
  BMP_array[36]=(BMP_payload_length >> 16) & 0xff; BMP_array[37]=(BMP_payload_length
      >> 24) & 0xff; //
  BMP_array[38]=0x00; BMP_array[39]=0x00;     BMP_array[40]=0x00; BMP_array[41]=0x00;
        // Print defs
  BMP_array[42]=0x00; BMP_array[43]=0x00;     BMP_array[44]=0x00; BMP_array[45]=0x00;
        //
  BMP_array[46]=0x02; BMP_array[47]=0x00;     BMP_array[48]=0x00; BMP_array[49]=0x00;
        // Number of colors in palette
  BMP_array[50]=0x00; BMP_array[51]=0x00;     BMP_array[52]=0x00; BMP_array[53]=0x00;
        // 0 important colors
  /* Color Palette */
  BMP_array[54]=0x00; BMP_array[55]=0x00;     BMP_array[56]=0x00; BMP_array[57]=0x00;
        // Black
  BMP_array[58]=0x00; BMP_array[59]=0x00;     BMP_array[60]=0xff; BMP_array[61]=0x00;
        // Red

  /* Filling DATA with hadamard line */
  for(int byte=0; byte<BMP_payload_length; byte++){
    BMP_array[OFFSET+byte]=0;
    b=byte*8;

    for(int bit=0; bit<8; bit++)
      BMP_array[byte+OFFSET]+=(pixel_map_array[b+bit]<<(7-bit));
  }

  return 0;
}

int Write_BMP_File(string bmpName, int val_length, double values[]){

  int height = sqrt(val_length);
  int width = height;
  int load_length = val_length*3;
  int NOFF = BMP_HEADER+DIB_HEADER;
  int total_length = NOFF+load_length;
  int8 * BMP_array = new int8[total_length];
```

```
double val;

ofstream imgFile;

/* Writing BMP HEADER */
BMP_array[0]=0x42; BMP_array[1]=0x4d;
                                                    // BM
BMP_array[2]=total_length & 0xff;           BMP_array[3]=(total_length >> 8) &
    0xff;    // BMP Size
BMP_array[4]=(total_length >> 16) & 0xff;   BMP_array[5]=(total_length >> 24) &
    0xff;   //
BMP_array[6]=0x00; BMP_array[7]=0x00;       BMP_array[8]=0x00; BMP_array[9]=0x00;
         // whatever
BMP_array[10]=0x36; BMP_array[11]=0x00;     BMP_array[12]=0x00; BMP_array[13]=0x00;
        // DATA offset
/*Filling DIB HEADER */
BMP_array[14]=0x28; BMP_array[15]=0x00;     BMP_array[16]=0x00; BMP_array[17]=0x00;
        // DIB HEADER SIZE
BMP_array[18]=width & 0xff;                 BMP_array[19]=(width >> 8) & 0xff;
            // Width
BMP_array[20]=(width >> 16) & 0xff;         BMP_array[21]=(width >> 24) & 0xff;
           //
BMP_array[22]=height & 0xff;                BMP_array[23]=(height >> 8) & 0xff;
           // Height
BMP_array[24]=(height >> 16) & 0xff;        BMP_array[25]=(height >> 24) & 0xff;
          //
BMP_array[26]=0x01; BMP_array[27]=0x00;
                                                    // Number of color planes
BMP_array[28]=0x18; BMP_array[29]=0x00;
                                                    // Number of bits per pixel
BMP_array[30]=0x00; BMP_array[31]=0x00;     BMP_array[32]=0x00; BMP_array[33]=0x00;
        // Compression

BMP_array[34]=load_length & 0xff;           BMP_array[35]=(load_length >> 8) &
    0xff;    // Data Size
BMP_array[36]=(load_length >> 16) & 0xff;   BMP_array[37]=(load_length >> 24) &
    0xff;   //
BMP_array[38]=0x00; BMP_array[39]=0x00;     BMP_array[40]=0x00; BMP_array[41]=0x00;
        // Print defs
BMP_array[42]=0x00; BMP_array[43]=0x00;     BMP_array[44]=0x00; BMP_array[45]=0x00;
        //
BMP_array[46]=0x00; BMP_array[47]=0x00;     BMP_array[48]=0x00; BMP_array[49]=0x00;
        // Number of colors in palette
BMP_array[50]=0x00; BMP_array[51]=0x00;     BMP_array[52]=0x00; BMP_array[53]=0x00;
        // 0 important colors

/* Filling Load with Values as gray scale */
for(int byte=0; byte<val_length; byte++){
  val = values[byte]*256;
  BMP_array[NOFF+byte*3]=val;
  BMP_array[NOFF+byte*3+1]=val;
```

```
    BMP_array[NOFF+byte*3+2]=val;
  }

  imgFile.open(bmpName);
  /* Writing to file */
  for(int i=0; i<total_length; i++)
    imgFile<<BMP_array[i];

  imgFile.close();

  delete [] BMP_array;

  return 0;
}
```

C.5 SHELF.h

In this section we present a library of functions created to be used in `DMD-CS.cpp`.

```
/*
 * SHELF.h. The present functions handle generation of non-repeating pseudo-random
 * integers and choices made by users when interacting with COSAC Control-PC Side.
 *
 * Copyright (C) 2018 Joao Rino Silvestre
 *
 * This program is free software; you can redistribute it and/or modify it under
 * the terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 2 of the License, or (at your option) any later
 * version.
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
 */

#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <unistd.h>
#include <string.h>
#include <time.h>
typedef unsigned char int8;
typedef unsigned short int16;

using namespace std;

/* Generation of random ordered indexes for lines to be fetched */
void Random_Lines(double seed, int k, int *Lines, int max){
  srand(seed);
  int rand_hold;
  bool is_rep;
  bool check_rep;

  for(int i=0; i<k; i++){
    check_rep=false;
    rand_hold=rand()%max;

    while(!check_rep){
      is_rep=false;

      for(int j=0; j<i && !is_rep; j++){
        if(rand_hold==Lines[j]){
          is_rep=true;
```

```
          rand_hold=rand()%max;
        }
      }
      if(!is_rep)
        check_rep=true;
    }
    Lines[i]=rand_hold;
  }
}


/* "Menu" type function, fed with a query string about a binary choice */
int Binary_Choice(string query){
  int choice=2;
  bool flag=0;

  while(choice!=0 && choice!=1){
    if(flag==1)                  // If an invalid choice is made, displays message
      cout<<"Invalid input"<<endl;

    cout<<query;
    cin>>choice;
    cout<<endl;
    flag=1;}

  return choice;
}
```

C.6 hadIndexToMatrix.cpp

In this section we present the code of a program created to convert a file containing the indices of Hadamard matrices rows used during a measurement into a file containing the elements of those rows.

```
/*
 * COSAC Hadamard-Indices-to-Matrix. The purpose of this program is to create a
 * CSV file holding the sampling matrix which vectors are rows of an Hadamard
 * matrix. The indices of these rows are held in the input file. This way COSAC's
 * users can use another reconstruction software to perform the signal recovery
 * task.
 *
 * Copyright (C) 2018 Joao Rino Silvestre
 *
 * This program is free software; you can redistribute it and/or modify it under
 * the terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 2 of the License, or (at your option) any later
 * version.
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
 */

#include <stdlib.h>
#include <cstring>
#include <iostream>
#include <fstream>
#include <math.h>
#include <cstdint>
#include "BMP.h"

using namespace std;

main(){

  int * indexes;
  int8 * had_Line;
  int * powers;
  string holdStr, indexName, matName;
  int had_Order, N, K;
  string prefix = "Mat_";
  ifstream indexFile;
  ofstream matFile;

  cout<<"This directory presently holds the following files:"<<endl<<endl;
  system("ls");
```

```
cout<<endl;
cout<<"If the file you wish to use to parse the indices from is present in this
    folder, please input its name; if not move the file into this folder and then
    input its name."<<endl;
cout<<"Please input the index file's name: "; cin>>indexName; cout<<endl<<endl;

matName = prefix+indexName;

indexFile.open(indexName);

if(!indexFile.is_open()){
  cout<<"There was an error attempting to open "<<indexName<<"."<<endl;
  return -1;
}

getline(indexFile,holdStr);
had_Order = stoi(holdStr);
K = 0;

while(getline(indexFile,holdStr))
  K++;

indexes = new int[K];
indexFile.close();

indexFile.open(indexName);
getline(indexFile,holdStr);

for(int i=0; getline(indexFile,holdStr); i++)
  indexes[i] = stoi(holdStr);

indexFile.close();

N = pow(2,had_Order);
had_Line = new int8 [N];
powers = new int[had_Order-1];
powers[0] = 1;

for(int p=1; p<had_Order; p++)
  powers[p] = powers[p-1]*2;

matFile.open(matName);

if(!matFile.is_open()){
  cout<<"There was an error attempting to open "<<matName<<"."<<endl;
  return -1;
}

for(int i=0; i<K; i++){

  Create_Hadamard_Lines(indexes[i],had_Order,had_Line,powers);
```

```
    for(int j=0; j<N; j++){
   matFile<<(int)had_Line[j];

      if(j!=(N-1))
   matFile<<';';
      else matFile<<endl;
    }
  }

  matFile.close();

  delete [] indexes; delete [] had_Line; delete [] powers;
}
```

C.7 pIDDO-10/24bit.ino Flow Chart

In this section we present a flow chart of the processes and interactions of the acquisition control programs, `pIDDO-10/24bit`, with the signal coding control program `DMD-CS.cpp` and DLP LightCrafter's FPGA.

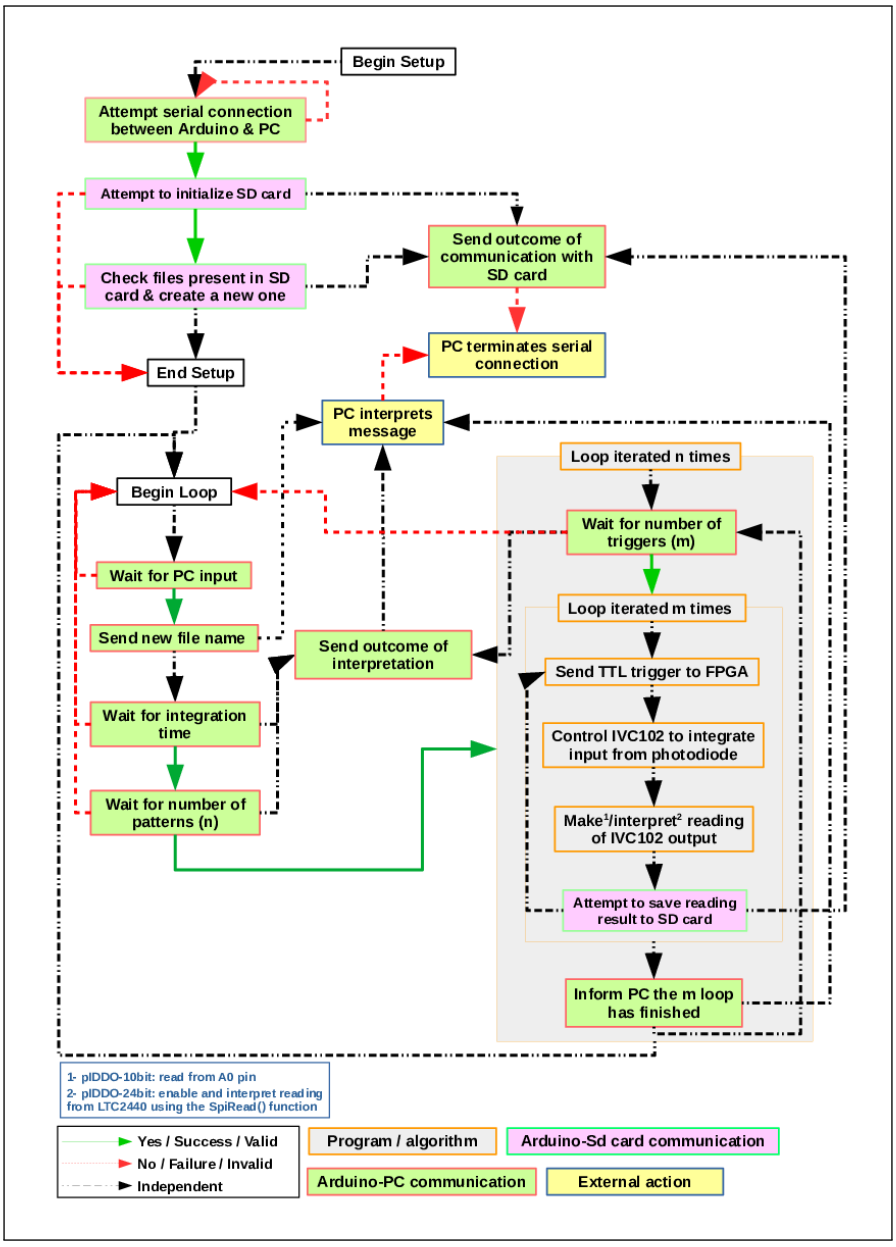


Figure C.2: *Simplified flowchart of tasks and interactions of the acquisition control program.*

C.8 pIDDO-10bit.ino

In this section we present the control program for the 10 bit version of the acquisition subsystem.

```
/*
 * COSAC Control-Arduino Side. The purpose of this program is to control IC's of
 * the pIDDO-10bit PCB and to communicate with DM365 in order to take measurements.
 *
 * Copyright (C) 2019 Joao Rino Silvestre
 *
 * This program is free software; you can redistribute it and/or modify it under
 * the terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 2 of the License, or (at your option) any later
 * version.
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
*/

#include <SPI.h>
#include <SD.h>
#include <string.h>

/* IVC */
const int S1 = 3; // IVC pin 11
const int S2 = 2; // IVC pin 12

/* SD */
const int CS_SD = 4; // /CS pin
const int CD_SD = 5;
File myFile;
String prefix = "RUN";
String suffix = ".CSV";
int nameNum = 0;
String fileName = prefix + String(nameNum) + suffix;

/* Trigger */
const int TRIG_OUT = 6; // for TTL communication with DM365
const int TRIG_IN = 7; // for TTL communication with DM365
char num;
int trigNum;
long patNum;
long curNum;

int avgA, avgB, outValue;
int nrpt = 10;
String dummy;
```

```
long int intTime;
bool delay_flag;

void setup() {

  Serial.begin(115200);    // set up serial comm to PC at this baud rate

  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(CS_SD, OUTPUT);
  pinMode(CD_SD, INPUT_PULLUP);
  pinMode(TRIG_OUT, OUTPUT);
  pinMode(TRIG_IN, INPUT);

  digitalWrite(CS_SD, HIGH); // selecting SD card
  digitalWrite(TRIG_OUT, HIGH);
  delay(1);
  digitalWrite(TRIG_OUT, LOW);

  SPI.begin(); // initialize SPI, covering SDO,SDI,SCK signals
  SPI.setBitOrder(MSBFIRST); // data is clocked in MSB first
  SPI.setDataMode(SPI_MODE0); // SCLK idle low (CPOL=0), SDO read on rising edge
      (CPHI=0)
  SPI.setClockDivider(SPI_CLOCK_DIV16); // system clock = 16 MHz, chip max = 1 MHz

  while(!Serial.available() || (Serial.available() && Serial.read()!='S')){}

  if (digitalRead(CD_SD)) {

    if (!SD.begin(CS_SD)) { // checks if SD is ready for communication
      Serial.print('N');
      Serial.flush();
      return;
    }

    Serial.print('Y');
    Serial.flush();

    for (int i = 1; SD.exists(fileName); i++) { //check for the existence of previous
        runs
      fileName = prefix + String(i) + suffix; //fileName+String(i)+".csv"
      nameNum = i;
    }

    if (digitalRead(CD_SD))
        myFile = SD.open(fileName, O_CREAT | O_WRITE);

  }
  else {
    Serial.print('N');
    Serial.flush();
```

```
  }

  digitalWrite(S2, HIGH); // active high, avoids double instruction on loop
  digitalWrite(S1, HIGH); //avoid repetition on loop
}

void loop() {

  avgA = 0;
  avgB = 0;
  num = '0';
  patNum = 0;
  curNum = 0;
  intTime = 0;
  delay_flag = false;

  while(!Serial.available() || (Serial.available() && Serial.read()!='F')){}

  fileName = prefix+String(nameNum)+suffix;
  Serial.print("?"+fileName+"!");
  Serial.flush();

  if (digitalRead(CD_SD))
    myFile = SD.open(fileName, O_CREAT | O_WRITE);

  while(!Serial.available() || (Serial.available() && Serial.read()!='?')){} //waits
      for input from PC for intTime
  delay(100);

  while(Serial.available()){
    num = Serial.read();
    delayMicroseconds(250);
    if(num >= '0' && num <= '9'){
      intTime = intTime*10+(num-'0');
    }
    else{
      break;
    }
  }

  if(!(num == '!')){
    delayMicroseconds(10);
    Serial.print('N');
    Serial.flush();
    return;
  }

  delayMicroseconds(10);
  Serial.print('Y');
  Serial.flush();
```

```
if(intTime > 10000){
  delay_flag = true;
  intTime /= 1000;
}

while(!Serial.available() || (Serial.available() && Serial.read()!='?')){} //waits
    for input from PC for patNum
delay(100);

while(Serial.available()){
  num = Serial.read();
  delayMicroseconds(250);
  if(num >= '0' && num <= '9'){
    patNum = patNum*10+(num-'0');
  }
  else{
    break;
  }
}

if(!(num == '!')){
  delayMicroseconds(10);
  Serial.print('N');
  Serial.flush();
  return;
}

delayMicroseconds(10);
Serial.print('Y');
Serial.flush();

while (curNum < patNum) {

  trigNum = 0;

  while(!Serial.available() || (Serial.available() && Serial.read()!='?')){} //waits
      for input from PC for trigNum
  delay(10);

  while(Serial.available()){
    num = Serial.read();
    delayMicroseconds(250);
    if(num >= '0' && num <= '9')
      trigNum = trigNum*10 + (num-'0');
    else
      break;
  }

  if(!(num == '!')){
    delayMicroseconds(10);
    Serial.print('N');
```

```
  Serial.flush();
  return;
}

curNum += trigNum;
num = '0';
delayMicroseconds(10);
Serial.print('Y');
Serial.flush();
delay(1000);

for(int i=0; i<trigNum; i++){ //sends triggers to FPGA to change DMD config;
                              //activates IVC to integrate current;
                              //reads conversion from ADC and norms it;
                              //saves resuls to file

  digitalWrite(TRIG_OUT, HIGH); // Triggers for config switch
  delayMicroseconds(20);
  digitalWrite(TRIG_OUT, LOW);

  digitalWrite(S2, LOW); //reset
  delayMicroseconds(50);
  digitalWrite(S2, HIGH); //pre integration hold & offset measurement
  delayMicroseconds(5);

  for(int i=0; i<nrpt; i++){
    avgA += analogRead(A0);
  }
  avgA /= nrpt;

  digitalWrite(S1, LOW);
  if(delay_flag)
    delay(intTime);
  else
    delayMicroseconds(intTime);

  digitalWrite(S1, HIGH); //reset prep

  for(int i=0; i<nrpt; i++){
    avgB += analogRead(A0);
  }
  avgB /= nrpt;
  outValue = avgB - avgA;
  if (digitalRead(CD_SD)) {
    if (myFile) {
      myFile.println(outValue);
    }
    else {
      delayMicroseconds(10);
      Serial.print('X');
      Serial.flush();
```

```
      }
    }
    else {
      delayMicroseconds(10);
      Serial.print('Z');
      Serial.flush();
    }
  }
  delayMicroseconds(10);
  Serial.print('T');
  Serial.flush();
 }
 myFile.close();
 nameNum++;
}
```

C.9 pIDDO-24bit.ino

In this section we present the control program for the 24 bit version of the acquisition subsystem.

```
/*
 * COSAC Control-Arduino Side. The purpose of this program is to control IC's of
 * the pIDDO-24bit PCB and to communicate with DM365 in order to take measurements.
 *
 * Copyright (C) 2010 John Beale
 * Copyright (C) 2019 Joao Rino Silvestre
 *
 * This program is free software; you can redistribute it and/or modify it under
 * the terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 2 of the License, or (at your option) any later
 * version.
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
*/

#include <SPI.h>
#include <SD.h>
#include <string.h>

/* IVC */
const int S1 = 3; // IVC pin 11
const int S2 = 2; // IVC pin 12

/* LTC */
const int CS_ADC = 9; // digital pin 11 for /CS
const int BUSY_ADC = 10; // digital pin 8 for BUSY readout
const byte cmd = 0x10; // command word to select conversion rate
// 0x78->6.9Hz; 0x48->13.8Hz; 0x40->27.5Hz
// 0x38->55Hz;  0x30->110Hz;  0x28->220Hz
// 0x20->440Hz; 0x18->880Hz;  0x00->880Hz;
// 0x10->1.76kHz; 0x08->3.52kHz;

/* SD */
const int CS_SD = 4; // /CS pin
const int CD_SD = 5;
File myFile;
String prefix = "RUN";
String suffix = ".CSV";
int nameNum = 0;
String fileName = prefix + String(nameNum) + suffix;

/* Trigger */
```

```
const int TRIG_OUT = 6; // for TTL communication with DM365
const int TRIG_IN = 7; // for TTL communication with DM365
char num;
int trigNum = 0;
long patNum = 0;
long curNum = 0;

/* Variables */
const long int baud_rate = 115200;
long out, out_A, out_B;
int nsample = 10;
String dummy;
long int intTime;
bool delay_flag;

void setup(){

  Serial.begin(baud_rate);

  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);

  pinMode(CS_ADC, OUTPUT);
  pinMode(BUSY_ADC, INPUT);

  pinMode(CS_SD, OUTPUT);
  pinMode(CD_SD, INPUT_PULLUP);

  pinMode(TRIG_OUT, OUTPUT);
  pinMode(TRIG_IN, INPUT);

  digitalWrite(CS_ADC, HIGH); // chip select is active low
  digitalWrite(CS_SD, HIGH); // selecting SD card
  digitalWrite(TRIG_OUT, HIGH);
  delay(1);
  digitalWrite(TRIG_OUT, LOW);

  SPI.begin(); // initialize SPI, covering MOSI,MISO,SCK signals
  SPI.setBitOrder(MSBFIRST); // data is clocked in MSB first
  SPI.setDataMode(SPI_MODE0); // SCLK idle low (CPOL=0), MOSI read on rising edge
      (CPHI=0)
  SPI.setClockDivider(SPI_CLOCK_DIV16); // system clock = 16 MHz, chip max = 1 MHz

  while(!Serial.available() || (Serial.available() && Serial.read()!='S')){}

  if (digitalRead(CD_SD)) {

    if (!SD.begin(CS_SD)) { // checks if SD is ready for communication
      Serial.print('N');
      Serial.flush();
      return;
```

```
    }

    Serial.print('Y');
    Serial.flush();

    for (int i = 1; SD.exists(fileName); i++) { //check for the existence of previous
        runs
      fileName = prefix + String(i) + suffix; //fileName+String(i)+".csv"
      nameNum = i;
    }

    if (digitalRead(CD_SD))
        myFile = SD.open(fileName, O_CREAT | O_WRITE);

  }
  else {
    Serial.print('N');
    Serial.flush();
  }

  digitalWrite(S2, HIGH); // active high, avoids double instruction on loop
  digitalWrite(S1, HIGH); //avoid repetition on loop
}

// ================================================================================

// Main Loop:
// acquire readings, convert to units of volts, and send out on serial port

void loop() {

  num = '0';
  patNum = 0;
  curNum = 0;
  intTime = 0;
  delay_flag = false;

  while(!Serial.available() || (Serial.available() && Serial.read()!='F')){}

  fileName = prefix+String(nameNum)+suffix;
  Serial.print("?"+fileName+"!");
  Serial.flush();

  if (digitalRead(CD_SD))
    myFile = SD.open(fileName, O_CREAT | O_WRITE);

  while(!Serial.available() || (Serial.available() && Serial.read()!='?')){} //waits
      for input from PC for intTime
  delay(100);

  while(Serial.available()){
```

```
  num = Serial.read();
  delayMicroseconds(250);
  if(num >= '0' && num <= '9'){
    intTime = intTime*10+(num-'0');
  }
  else{
    break;
  }
}

if(!(num == '!')){
  delayMicroseconds(10);
  Serial.print('N');
  Serial.flush();
  return;
}

delayMicroseconds(10);
Serial.print('Y');
Serial.flush();

if(intTime > 10000){
  delay_flag = true;
  intTime /= 1000;
}

while(!Serial.available() || (Serial.available() && Serial.read()!='?')){} //waits
    for input from PC for patNum
delay(100);

while(Serial.available()){
  num = Serial.read();
  delayMicroseconds(250);
  if(num >= '0' && num <= '9'){
    patNum = patNum*10+(num-'0');
  }
  else{
    break;
  }
}

if(!(num == '!')){
  delayMicroseconds(10);
  Serial.print('N');
  Serial.flush();
  return;
}

delayMicroseconds(10);
Serial.print('Y');
Serial.flush();
```

```
while (curNum < patNum) {

  trigNum = 0;

  while(!Serial.available() || (Serial.available() && Serial.read()!='?')){} //waits
      for input from PC for trigNum
  delay(100);

  while(Serial.available()){
    num = Serial.read();
    delayMicroseconds(250);
    if(num >= '0' && num <= '9')
      trigNum = trigNum*10 + (num-'0');
    else
      break;
  }

  if(!(num == '!')){
    delayMicroseconds(10);
    Serial.print('N');
    Serial.flush();
    return;
  }

  curNum += trigNum;
  num = '0';
  delayMicroseconds(10);
  Serial.print('Y');
  Serial.flush();
  delay(1000);

  for(int i=0; i<trigNum; i++){ //sends triggers to FPGA to change DMD config;
                                //activates IVC to integrate current;
                                //reads conversion from ADC and norms it;
                                //saves resuls to file

    digitalWrite(TRIG_OUT, HIGH); // Triggers for config switch
    delayMicroseconds(20);
    digitalWrite(TRIG_OUT, LOW);

    digitalWrite(S2, LOW); //reset
    delayMicroseconds(200);
    digitalWrite(S2, HIGH); //pre integration hold & offset measurement

    out_A=0;
    for(int j=0; j<nsample; j++){
       while (digitalRead(BUSY_ADC));
       out_A += SpiRead();
    }
    out_A /= nsample;
```

```
      digitalWrite(S1, LOW); //integration hold

      if(delay_flag)
        delay(intTime);
      else
        delayMicroseconds(intTime);

      digitalWrite(S1, HIGH); //reset prep & signal measurement

      out_B=0;
      for(int j=0; j<nsample; j++){
         while (digitalRead(BUSY_ADC));
         out_B += SpiRead();
      }
      out_B /= nsample;

      out = out_B-out_A;

      if (digitalRead(CD_SD)) {
        if (myFile) {
          myFile.println(out);
        }
        else {
          delayMicroseconds(10);
          Serial.print('X');
          Serial.flush();
        }
      }
      else {
        delayMicroseconds(10);
        Serial.print('Z');
        Serial.flush();
      }
    }
    delayMicroseconds(10);
    Serial.print('T');
    Serial.flush();
  }
  myFile.close();
  nameNum++;
}
```

C.10 readOutLTC.ino

In this section we present a program used to test SPI communication with the LTC2440.

```
/*
 * LTC2440 readout program. The purpose of this program is to
 * output to a monitor the conversions performed by an LTC2440.
 * This program is based on code provided by the user jbeale from the
 * DangerousPrototypes.com forum.
 *
 * Copyright (C) 2010 John Beale
 * Copyright (C) 2018 Joao Rino Silvestre
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
 */
#include <SPI.h>

/* LTC */
const int CS_ADC = 9; // digital pin 11 for /CS
const int BUSY_ADC = 10; // digital pin 8 for BUSY readout
const byte cmd = 0x30; // 0x78->6.9Hz; 0x48->13.8Hz; 0x40->27.5Hz
// 0x38->55Hz;  0x30->110Hz;  0x28->220Hz
// 0x20->440Hz; 0x18->880Hz;  0x00->880Hz;
// 0x10->1.76kHz; 0x08->3.52kHz;

/* variables */
long volts_A, volts_B;
long volts;
double volts_d;
const long in_max = 16777215;
const double v_max = 2.5;
int n = 10;

void setup() {

  Serial.begin(115200);    // set up serial comm to PC at this baud rate

  pinMode(CS_ADC, OUTPUT);
  pinMode(BUSY_ADC, INPUT);
```

```
  digitalWrite(CS_ADC, HIGH); // chip select is active low

  SPI.begin(); // initialize SPI, covering MOSI,MISO,SCK signals
  SPI.setBitOrder(MSBFIRST); // data is clocked in MSB first
  SPI.setDataMode(SPI_MODE0); // SCLK idle low (CPOL=0), MOSI read on rising edge
      (CPHI=0)
  SPI.setClockDivider(SPI_CLOCK_DIV16); // system clock = 16 MHz, chip max = 1 MHz
}


// ============================================================================

// Main Loop:
// acquire readings, convert to units of volts, and send out on serial port

void loop() {

  volts_A=0;
  for(int j=0; j<n; j++){
    while (digitalRead(BUSY_ADC));
    volts_A += SpiRead()/n;
  }

  delay(1000);

  volts_B=0;
  for(int j=0; j<n; j++){
    while (digitalRead(BUSY_ADC));
    volts_B += SpiRead()/n;
  }

  volts = volts_B-volts_A;
  volts_d = ((double) volts/in_max)*v_max;
  Serial.print("B-A: ");
  Serial.println(volts_d,3);
}
```

C.11 LinearityTest.ino

In this section we present a program used to test the linearity of the LTC2440.

```
/*
 * LTC2440 Linearity Test program. The purpose of this program is to
 * make acquisitions with progressively more integration time,
 * increasing the output EPD (assuming constant lighting conditions),
 * and registering the output of the LTC. The collected data can then
 * be graphed in order to evaluate the linearity of the ensemble.
 * This program is based on code provided by the user jbeale from the
 * DangerousPrototypes.com forum.
 *
 * Copyright (C) 2010 John Beale
 * Copyright (C) 2018 Joao Rino Silvestre
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
 */

#include <SPI.h> //include the SPI library

const int S1 = 3; //IVC S1 pin 11
const int S2 = 2; //IVC S2 pin 12
const byte CS = 9; //LTC /CS pin 11
const byte busyPin = 10; //LTC BUSY pin 15

const long in_max=16777215;
const double v_max=2.5;

double volts;
long in; // incoming serial 32-bit word

int i = 1;
int xTime=50;
int pTime=50;
String seriesName = "Test ";

const byte cmd = 0x00; // 0x78->6.9Hz; 0x48->13.8Hz; 0x40->27.5Hz
```

```
                        // 0x38->55Hz; 0x30->110Hz; 0x28->220Hz;
                        // 0x20->440Hz; 0x18->880Hz; 0x00->880Hz;
                        // 0x10->1.76kHz; 0x08->3.52kHz;

void setup() {

  Serial.begin(115200); // set up serial comm to PC at this baud rate

  Serial.flush();
  Serial.println(seriesName+String(i));
  Serial.println(" ");

  pinMode(S1,OUTPUT);
  pinMode(S2,OUTPUT);
  pinMode(CS,OUTPUT);
  pinMode(busyPin,INPUT);
  digitalWrite(CS,HIGH); //chip select is active low
  digitalWrite(S2,HIGH); //active high, avoids double instruction on loop
  digitalWrite(S1,HIGH); //avoid repetition on loop

  SPI.begin(); //initialize SPI, covering MOSI,MISO,SCK signals
  SPI.setBitOrder(MSBFIRST); //data is clocked in MSB first
  SPI.setDataMode(SPI_MODE0); //SCLK idle low(CPOL=0),
                              //MOSI read on rising edge(CPHI=0)
  SPI.setClockDivider(SPI_CLOCK_DIV16); //system clock = 16 MHz, chip max = 1 MHz
}

void loop() {

  if(volts >= v_max){
    volts = 0;
    xTime = pTime;
    i++;
    Serial.flush();
    Serial.println(" ");
    Serial.println(seriesName+String(i));
    Serial.println(" ");
  }


  digitalWrite(S2,LOW); //reset
  delay(250);
  digitalWrite(S2,HIGH); //pre-integration hold & offset measurement
  delay(250);
  digitalWrite(S1,LOW);
  delay(xTime);
  digitalWrite(S1,HIGH); //reset prep

  while(digitalRead(busyPin)); //hold while LTC's Busy pin is High

  in = SpiRead();
```

```
  volts = in/((double)in_max)*v_max; //converts the binary to a readable measurement

  Serial.print(xTime);
  Serial.print(", ");
  Serial.println(volts,6);

  xTime += pTime;

} // end main loop
```

C.12 SpiRead()

In this section we present a function used to communicate with the LTC2440.

```
long SpiRead(void) {

  long result = 0x00000000;
  int b = 0x00;

  digitalWrite(CS_ADC, LOW); // take the SS pin low to select the chip

  b = SPI.transfer(cmd); // B3
  result = b << 8;
  b = SPI.transfer(cmd); // B2
  result |= b;
  result = result << 8;
  b = SPI.transfer(cmd); // B1
  result |= b;
  result = result << 8;
  b = SPI.transfer(cmd); // B0
  result |= b;

  digitalWrite(CS_ADC, HIGH);

  result &= 0x1fffffff; // force high four bits to zero

  result = result >> 5; // truncate lowest 5 bits

  return (result);
}
```

C.13 SimulateObservation.m

In this section we present a program used to simulate measurements performed by a CS camera.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Code to simulate an observation using the DMD and the KIDDO acquisition board
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Copyright (C) 2015 Alberto Krone-Martins (algol@sim.ul.pt)
% Copyright (C) 2015 Raquel Bandarra Borges (raquelbb@fc.ul.pt)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    This program is free software: you can redistribute it and/or modify
%    it under the terms of the GNU General Public License as published by
%    the Free Software Foundation, either version 3 of the License, or
%    (at your option) any later version.
%
%    This program is distributed in the hope that it will be useful,
%    but WITHOUT ANY WARRANTY; without even the implied warranty of
%    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
%    GNU General Public License for more details.
%
%    You should have received a copy of the GNU General Public License
%    along with this program. If not, see <http://www.gnu.org/licenses/>.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Given:
%        - An integration time (in seconds)
%        - A number of patterns
%        - A number samples
%        - The amplification factor (number of DMD micromirrors per sample)
%        - The ADC gain (ADUs/electrons)
%        - The additive gaussian noise level (in ADUs) ::: NOT IMPLEMENTED
%        - The background level (in ADU/s)         ::: NOT IMPLEMENTED
%        - The flux of the pixel of maximal flux (electron/s)
%        - The image filename (the image is considered to be normalized to
%        the maximum pixel; if it is not, the code will normalize it anyway)
%
% Performs:
%        Creates the Patterns based on a Hadamard matrix, simulate the observation
% of a certain signal (image) using the each of the Patterns and a KIDDO integration
% board with a Photodiode (considering the Poisson noise from the photon arrival
% process, the additive gaussian noise from electronics and background signal)
%
% Returns:
%        - A vector with the observed measurements
%        - The sampling matrix
%
%
% Usage example:
%
%   - Run the simulator
%   SimulateObservation(1, 4, 64, 8, 'HDF-South-close_64x64.tiff');
```

```
%
%   The above parameters mean that the acquisition will be performed using
%   a 1 second integration time, there will be 4 aquisitions of different
%   patterns, each pattern will be a 64x64 matrix and each sample will be composed
%   by 8x8 micromirrors of the DMD. The generator will use the L
%   lightcrafter object and the tcpObject.
%
%
% Example filenames ::
%                     'HDF-South-close_64x64.tiff'
%                     'MariaJoaoPires64.tiff'
%
% [obV, sMat] = SimulateObservation(1, 10, 64, 1, 1, 200, 0, false, 1000,
    'HDF-South-close_64x64.tiff');
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [obsVec, samplingMat] = SimulateObservation(intTime, nPatterns, nSamples,
    gFactor, gain, gaussNoiseStDev, backgroundLevelADU, poissonNoise, fluxAtMaxPix,
    filename)

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Input information
    n = nPatterns;                % Number of measurements to perform
    dmdX = 684;                   % This is the Lightcrafter standard size...
    dmdY = 608;                   % This is the Lightcrafter standard size...
    obsVec = zeros(nPatterns,1);  % Create the observational vector (initially empty)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    inimage = imread(filename); % Open the image file
    inimage = inimage(:,:,1);  % Get only the data that matters...
    inimage = double((inimage - min(inimage(:)))) ./
        double((max(inimage(:)-min(inimage(:)))));
    inimage = inimage .* fluxAtMaxPix;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    disp('Preparing the sampling matrix...');
    totNpix = nSamples * nSamples;
    idx = randperm(totNpix);                  % create the random indexes
    hadSamplingMat = hadamard(totNpix);       % create a hadamard matrix
    hadSamplingMat = hadSamplingMat(idx(1:n),:); % get a random subset of the matrix
    hadSamplingMat = hadSamplingMat + abs(min(min(hadSamplingMat))); % Normalization
    hadSamplingMat = hadSamplingMat/max(max(hadSamplingMat)); % Normalization
    samplingMat = double(hadSamplingMat);     % convert to double...
    imHad = zeros(684, 608, 3);
    disp('Preparing the sampling matrix... Done!');

    % The following loop will loop through the patterns,
    % at each pattern it will send the pattern to the DMD, and
    % perform an acquisition using the KIDDO board.
    disp('Performing the sampling...');
    for iPat = 1:nPatterns
```

```
fprintf('   Sample %i of %i...\n', iPat, nPatterns);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DMD pattern part
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Here, we reshape the iPat-th row of the Hadamard matrix into 2D matrix
disp('       Generating the pattern...');
pp = reshape(samplingMat(iPat,:), [nSamples, nSamples]);

% Here we multiply the pattern and an image so we can mock that a signal is
    being
% projected on top of the DMD array
bb = double(inimage);
%bb = bb/max(max(bb));
pp = pp.*bb;

% The following loop will fill the 3D matrix of a RGB bitmap that
% will be sent to the DMD using the MATLAB framework from Jan Winter
% There is very, very likely, a much smater way to do this in
% MATLAB... and even to communicate with the Lightcrafter without
% having to create a BMP file first, but well, this is working for
% the moment...
for i = 1:dmdX
    for j = 1:dmdY
        ic = ceil(i/gFactor);
        ij = ceil(j/gFactor);
        if ic <= nSamples
            if ij <= nSamples
                if poissonNoise
                   imHad(i, j, :) = poissrnd( pp( ic, ij ) * fluxAtMaxPix *
                       intTime, 3, 1);%*0.1 ;
                   imHad(i, j, 1) = poissrnd( pp( ic, ij ) * fluxAtMaxPix *
                       intTime );%*0.1 ;
                   imHad(i, j, 2) = poissrnd( pp( ic, ij ) * fluxAtMaxPix *
                       intTime );%*0.1 ;
                   imHad(i, j, 3) = poissrnd( pp( ic, ij ) * fluxAtMaxPix *
                       intTime );%*0.1 ;
                else
                   imHad(i, j, 1) = pp( ic, ij ) ;%*0.1 ;
                   imHad(i, j, 2) = pp( ic, ij ) ;%*0.1 ;
                   imHad(i, j, 3) = pp( ic, ij ) ;%*0.1 ;
                end
            end
        end
    end
end

imHad = uint16(imHad * intTime);
if poissonNoise
    imHad = imnoise(imHad, 'poisson');
end
```

```
%       imshow(imHad)
        % Now, send the pattern to the DMD
        disp('       Sending the pattern to the DMD...');
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Signal acquisition part
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %pause(0.1); % Sleep for 100 ms, just to make sure the DMD has set the new BMP

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        disp('       Acquiring the reading from the KIDDO board...');
        obsVec(iPat) = sum(sum(imHad(:,:,1)));

        %%%%

        %%%% Add Background at the KIDDO board level

        %%%% Add Gaussian readout noise at the KIDDO board level
        obsVec(iPat) += random('norm',0,gaussNoiseStDev);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        fprintf('   Sample %i of %i...Done!\n', iPat, nPatterns);


    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Save the sampling matrix
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    disp('Saving data to files...');
    dlmwrite('res-samplingMat.txt',samplingMat,';');
    dlmwrite('res-obsVec.txt',obsVec,'delimiter',';','precision','%.f');
    disp('Saving data to files... Done!');
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % That is is folks!
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    disp('Performing the sampling... Done!');
end
```

C.14 ReconstructImageFromFiles.m

In this section we present a template script used to apply the `ReconstructImage.m` function to data present on a file.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Code to reconstruct an image from a vector of observations and a sampling matrix
% This code was built for the simulations performed for RBB MSc Dissertation studies
% NOTE: This is just a wrapper for the ReconstructImage.m code, to provide
% easy access to files containing the registered observations and the sampling matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Copyright (C) 2015 Alberto Krone-Martins (algol@sim.ul.pt)
% Copyright (C) 2015 Raquel Bandarra Borges (raquelbb@fc.ul.pt)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    This program is free software: you can redistribute it and/or modify
%    it under the terms of the GNU General Public License as published by
%    the Free Software Foundation, either version 3 of the License, or
%    (at your option) any later version.
%
%    This program is distributed in the hope that it will be useful,
%    but WITHOUT ANY WARRANTY; without even the implied warranty of
%    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
%    GNU General Public License for more details.
%
%    You should have received a copy of the GNU General Public License
%    along with this program. If not, see <http://www.gnu.org/licenses/>.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Given:
%        - A CSV file with a vector with the observations
%        - A CSV file with a sampling matrix
%
% Performs:
%        - Reconstructs an image using L1 or TV optimization
%
% Returns:
%        - An array with the reconstructed image
%
%
% Usage example:
%
%   - Run the reconstructor
%   a = ReconstructImageFromFiles('res-obsVec.txt', 'res-samplingMat.txt',
    'HDF-South-close_64x64-recons.tiff',
%       'HDF-South-close_64x64-recons.txt');
%
%   The above parameters mean that the software will reconstruct the image
%   from the res-obsVec.txt file that stores the observations,
%   the res-samplingMat.txt file that stores the sampling matrix,
%   and will finally store the reconstructed image at the
%   HDF-South-close_64x64-recons.tiff file.
```

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [a] = ReconstructImageFromFiles(obsVecFilename, sampMatFilename,
    outImageFilename, outDataFilename)

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Load the files
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    disp('Image reconstruction from the CS Camera Prototype... reading files...');
    disp('    Reading observation file...');
    obV = dlmread(obsVecFilename, ',');
    disp('    Reading observation file... Done!');
    disp('    Reading sampling Matrix file...');
    sampMat = dlmread(sampMatFilename, ',');
    disp('    Reading observation file... Done!');
    disp('Image reconstruction from the CS Camera Prototype... reading files...Done!');

    %%%%%%%%%%
    %resample%
    %%%%%%%%%%
    sizeObV = size(obV);
    lengthObV = sizeObV(1);
    sizeMat = size(sampMat);
    colMat = sizeMat(2);

    newLength = floor(lengthObV*frac);
    oldIndexes = randsample(lengthObV,newLength);

    newObV = zeros(newLength,1);
    newSampMat = zeros(newLength,colMat);

    for i=1:newLength
        newObV(i)=obV(oldIndexes(i));
        newSampMat(i,1:colMat) = sampMat(oldIndexes(i),1:colMat);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Run the reconstruction
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    a = ReconstructImage(newObV, newSampMat, outImageFilename, outDataFilename);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % That is all folks!
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
```

C.15 ReconstructImage.m

In this section we present a function used to reconstruct CS measurements using one of six possible functions.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Code to reconstruct an image from a vector of observations and a sampling matrix
% This code was built for the simulations performed for RBB MSc Dissertation studies
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Copyright (C) 2015 Alberto Krone-Martins (algol@sim.ul.pt)
% Copyright (C) 2015 Raquel Bandarra Borges (raquelbb@fc.ul.pt)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    This program is free software: you can redistribute it and/or modify
%    it under the terms of the GNU General Public License as published by
%    the Free Software Foundation, either version 3 of the License, or
%    (at your option) any later version.
%
%    This program is distributed in the hope that it will be useful,
%    but WITHOUT ANY WARRANTY; without even the implied warranty of
%    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
%    GNU General Public License for more details.
%
%    You should have received a copy of the GNU General Public License
%    along with this program. If not, see <http://www.gnu.org/licenses/>.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Given:
%        - A vector with the observations
%        - A sampling matrix
%
% Performs:
%        - Reconstructs an image using L1 or TV optimization
%
% Returns:
%        - An array with the reconstructed image
%
% Usage example:
%
%   - Run the reconstructor
%   a = ReconstructImage(obV, sMat, 'HDF-South-close_64x64-recons.tiff');
%
%   The above parameters mean that the software will reconstruct the image
%   from the obv vector that stores the observations (measurements of the
%   acquisition system), the sMat (the sampling matrix that was created at
%   the DMD: each row is equivalent to one entire pattern) and finally
%   store the reconstructed image at the HDF-South-close_64x64-recons.tiff
%   file.
%
% [recImage] = ReconstructImage(obV, sMat, 'HDF-South-close_64x64-recons.bmp',
    'HDF-South-close_64x64-recons.txt');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%function ReconstructImage(obsVecFilename, sampMatFilename, outImageFilename,
```

```
    outDataFilename)
function [recIm] = ReconstructImage(obsVec, sampMat, outImageFilename,
    outDataFilename)

    disp('Image reconstruction from the CS Camera Prototype... Starting!');

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Configure necessary paths for the required mathematical libraries
    disp('    Configuring the environment for the reconstruction...');
    path(path, '~/Desktop/l1magic-1.11/l1magic');
    path(path, '~/Desktop/l1magic-1.11/l1magic/Optimization');
    path(path, '~/Desktop/GPSR_6.0');
    disp('    Configuring the environment for the reconstruction... Done!');
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Setup and solve the l1 of TV optimization problem
    disp('    Setting up and solving the TV optimization problem...');
    x0 = sampMat' * obsVec;                                    % Our problem is simple!
    tic                                                        % Just to measure the time
    %xp = l1eq_pd(x0, sampMat, [], obsVec, 1e-3);              % one can adopt
    %[xp,status] = l1_ls(sampMat,obsVec,5);                    % an L1 optimization...
    %[xp,xpd] = GPSR_BB(obsVec,sampMat,2);                     % a gradient projection
    %[xp,xpd] = GPSR_Basic(obsVec,sampMat,5);
    %xp = l1decode_pd(x0, sampMat, [], obsVec, 1e-1);          % l1 opt with noise
    xp = tvqc_logbarrier(x0, sampMat, [], obsVec, 1, 1e-2, 5); % or the TV optimization
    toc                                                        % Just to measure the time
    disp('    Setting up and solving the TV optimization problem... Done!');
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Generate some graphical output
    disp('    Generating some graphical output...');
    sampledSize = size(sampMat);                   % Get the image dimensions
    reconsImageDim = sqrt(sampledSize(2));

    recIm = reshape(xp,reconsImageDim,[]);
    imshow(recIm)

    normRecIm = reshape(xp,reconsImageDim,[])./max(xp);
    %imshow(normRecIm)

    imwrite(normRecIm, outImageFilename);
    dlmwrite(outDataFilename,normRecIm,'delimiter',';','precision','%g');
    disp('    Generating some graphical output... Done!');
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % That is all folks!
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    disp('Image reconstruction from the CS Camera Prototype... Done!');
```

```
end
```

C.16 GetSNRFromFolderFiles.m

In this section we present a template script written to facilitate the application of the `IterSignalNoiseAvg.m` function to data present in multiple files.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The function browses for txt files in folder path provided by the user, then %
% creates an output file named by the user, it parses the files into the %
% IterSignalNoiseAvg and uses its outputs to estimate the SNR of those of the %
% values in that value and also estimates that SNR error. SNR and its error are %
% then printed to the output file.                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Copyright (C) 2019 Joao Rino Silvestre (fc43510@alunos.fc.ul.pt)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   This program is free software: you can redistribute it and/or modify
%   it under the terms of the GNU General Public License as published by
%   the Free Software Foundation, either version 3 of the License, or
%   (at your option) any later version.
%
%   This program is distributed in the hope that it will be useful,
%   but WITHOUT ANY WARRANTY; without even the implied warranty of
%   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
%   GNU General Public License for more details.
%
%   You should have received a copy of the GNU General Public License
%   along with this program. If not, see <http://www.gnu.org/licenses/>.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [] = GetSNRFromFolderFiles(folderPath,outFileName)

    path = strcat(folderPath,'*.txt');

    files = dir(path);

    outFile = fopen(outFileName,'wt');

    fprintf(outFile,'Filename AvgSNR StdSNR\n');

    for file = files'
        mapName = file.name;
        [N,eN,S,eS]=IterSignalNoiseAvg(strcat(folderPath,mapName),.02);
        SNR = S/N;
        eSNR = eS/S;sqrt((eS/N)^2+(S*eN/(N^2))^2);

        fprintf(outFile,'%s %.5f %.5f\n',mapName,SNR,eSNR);

    end

    fclose(outFile);
end
```

C.17 IterSignalNoiseAvg.m

In this section we present a function written to extract an estimate of the average and standard deviation of noise and signal values presend in a reconstruction data file.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function employs an iterative 3sigma rejection algorithm to sort the %
% values in the file obMatName into two sets, Noise and Signal, the iterations %
% are processed until the relative difference between two consecutive average %
% noise values is inferior to the given epsilon.                              %
% The function returns the average noise (AvgN), its standard deviation (StdN) %
% the average signal (AvgS), its standard deviation (StdS), and the ration of %
% AvgS and AvgN (SNR).                                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Copyright (C) 2019 Joao Rino Silvestre (fc43510@alunos.fc.ul.pt)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    This program is free software: you can redistribute it and/or modify
%    it under the terms of the GNU General Public License as published by
%    the Free Software Foundation, either version 3 of the License, or
%    (at your option) any later version.
%
%    This program is distributed in the hope that it will be useful,
%    but WITHOUT ANY WARRANTY; without even the implied warranty of
%    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
%    GNU General Public License for more details.
%
%    You should have received a copy of the GNU General Public License
%    along with this program. If not, see <http://www.gnu.org/licenses/>.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [AvgN,StdN,AvgS,StdS,SNR] = IterSignalNoiseAvg(obMatName, epsilon)

    obMat = dlmread(obMatName,';'); %read value map
    obV = obMat(:);              %parse map to a column array
    obSize = size(obV);
    obLength = obSize(1);        %get array length

    obV = obV + (-1)*min(obV);   %make min(obV) = 0
    obV = obV / max(obV);        %normalize obV

    AvgN = mean(obV);
    StdN = std(obV);
    MaxN = AvgN+3*StdN;          %stipulate boundary

    r = Inf;
    iter = 0;

    while r>epsilon

        count = 0;
```

```
        for i=1:obLength
             if obV(i)>MaxN
                 count = count+1;
             end
        end

        tempN = zeros(obLength-count,1);
        tempS = zeros(count,1);
        countN = 1;
        countS = 1;

        for i=1:obLength
            if obV(i)>MaxN
                tempS(countS)=obV(i);
                countS = countS+1;
            else
                tempN(countN)=obV(i);
                countN = countN+1;
            end
        end

        newAvgN = mean(tempN);
        r = abs(AvgN-newAvgN)/AvgN;
        AvgN = newAvgN;
        StdN = std(tempN);
        MaxN = AvgN + 3*StdN;
        AvgS = mean(tempS);
        StdS = std(tempS);

        iter = iter +1;

        clearvars tempN tempS;

    end
end
```

Appendix D

Mechanical Support and Enclosure

In this appendix we present a list of parts produced for the optical and the structural subsystems and casing for COSAC, as well as their schematic drawings. Later we also present results for FEM simulations of the structural subsystem designed in this dissertation.

D.1 List of Produced Parts

In this section we present a list of components developed for the optical and the structural subsystems and casing for COSAC.

D.1.1 Photopolymer Resin Parts

In this section we present a list of parts produced using a stereolithographic 3d printer.

- 1× Bottom support for Hamamatsu S1223
- 1× Bottom support for ∅25.4mm, 4mm thick lens
- 1× Bottom support for two ∅25.4mm, 4mm & 6mm thick lens
- 6× Stopper for slide carriage
- 1× Top support for Hamamatsu S1223
- 2× Top support for a ∅25.4mm, 4mm thick lens
- 1× Top support for a ∅25.4mm, 6mm thick lens

D.1.2 Aluminum Parts

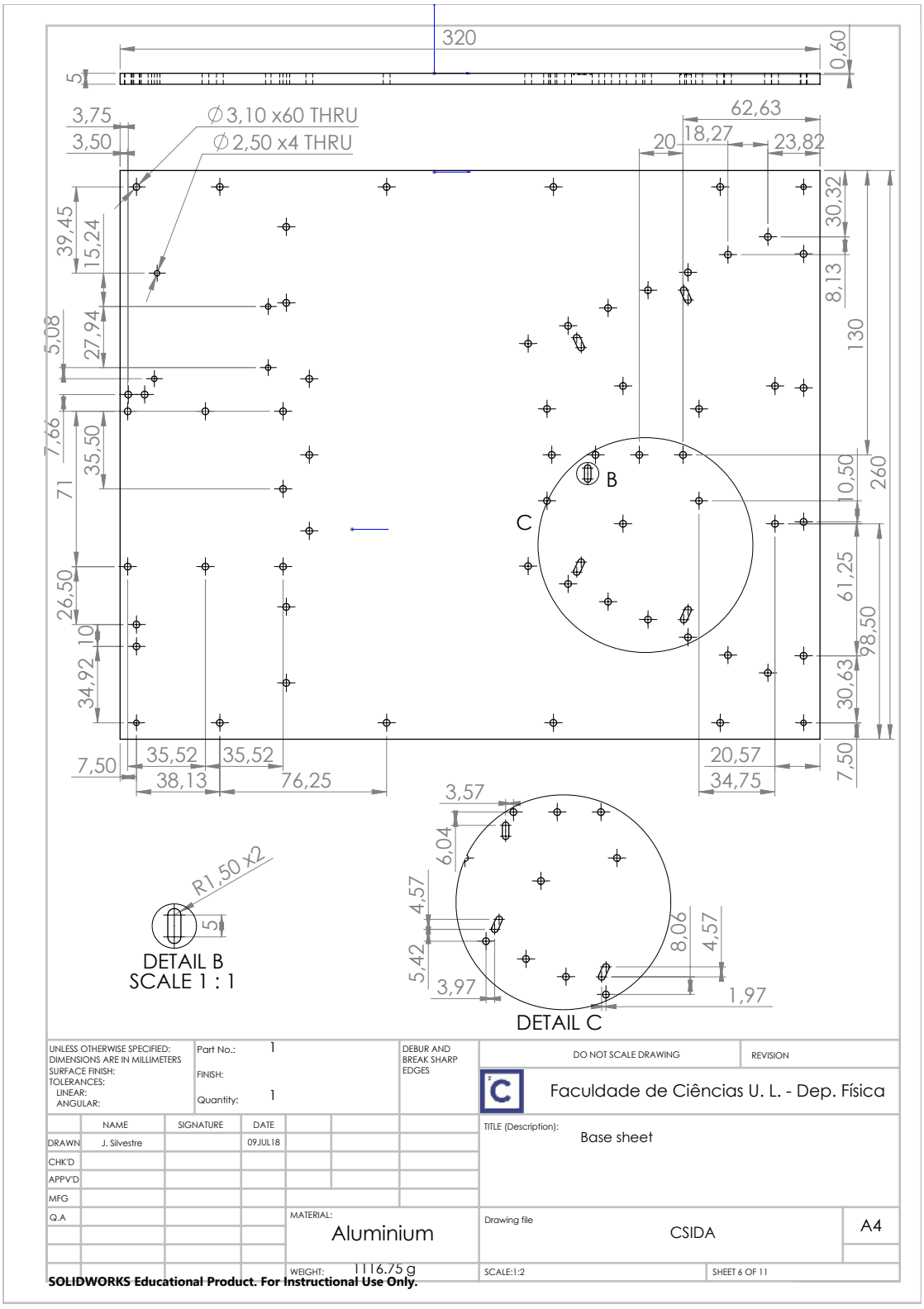
In this section we present a list of parts manufactured using a CNC and milling machines.

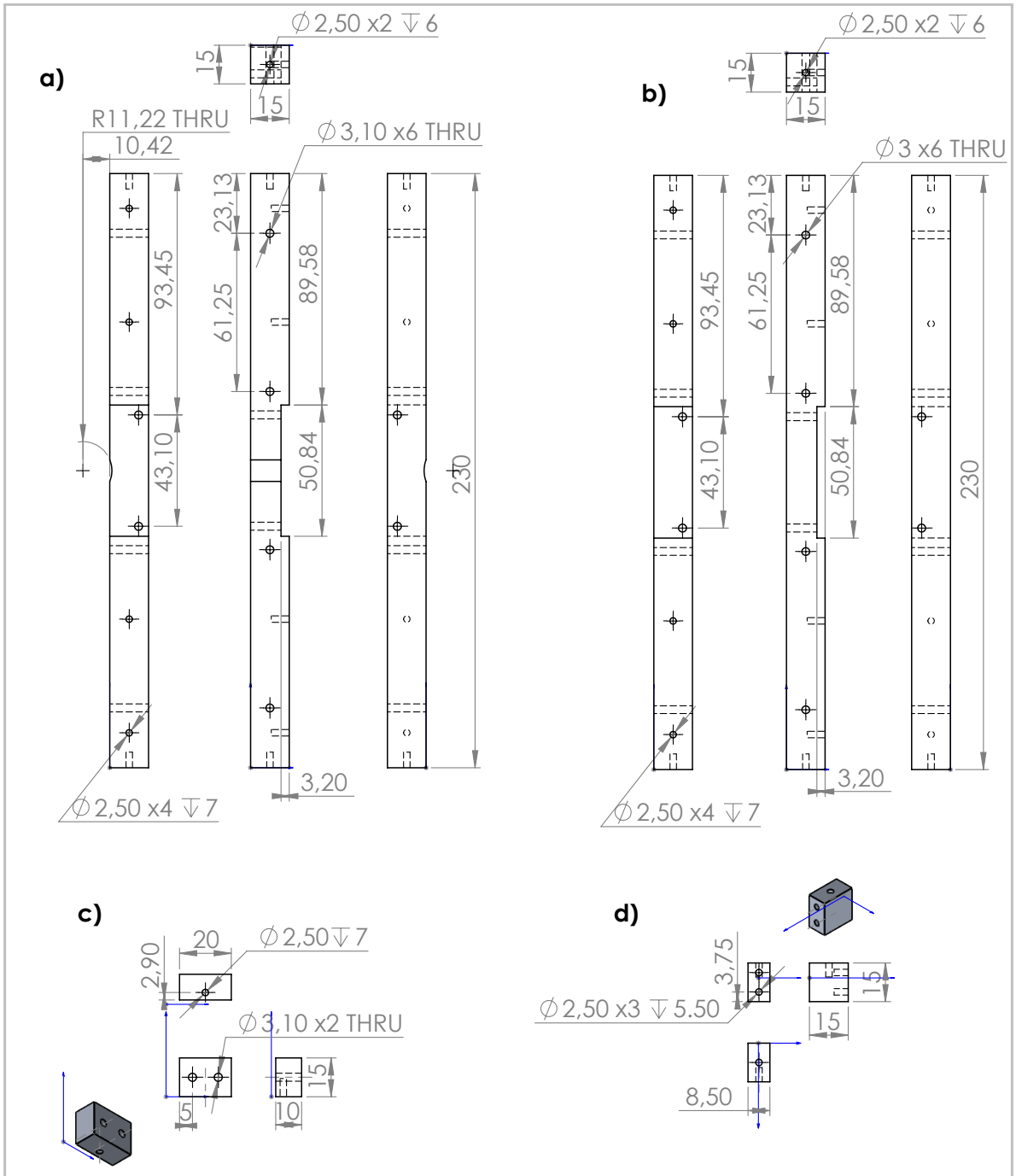
- 1× Back panel
- 1× Base
- 1× Cover panel
- 1× Flange

- 1× Front bottom beam
- 1× Front panel
- 1× Front top beam
- 1× Holder nº 1
- 1× Holder nº 2
- 2× Side panel
- 1× Support for DLP Lightcrafter

D.2 Components' Schematics

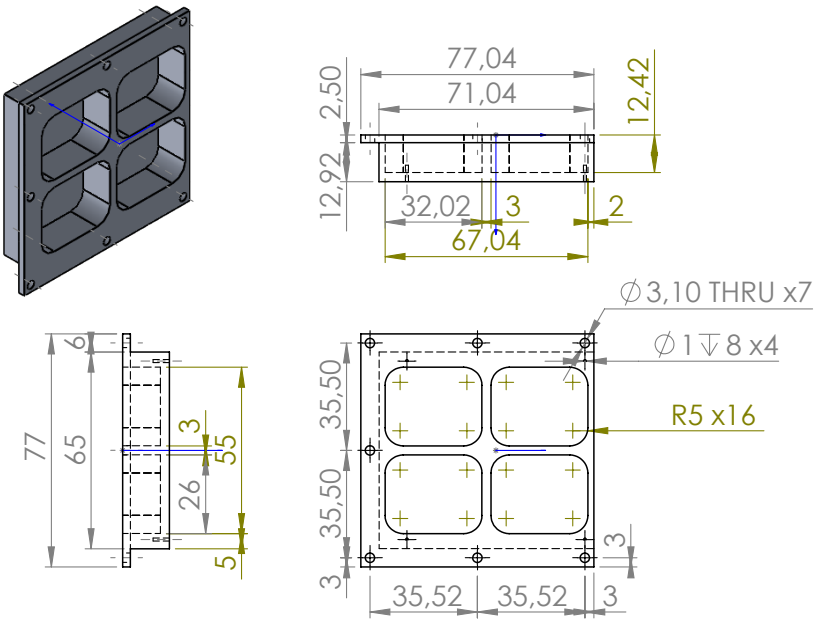
D.2.1 Aluminum Components



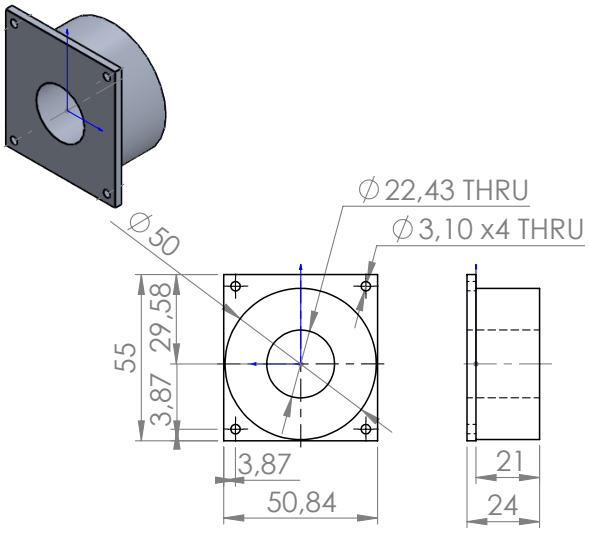


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	Part No.: a) 2, b) 3, c) 4, d) 5 FINISH: Quantity: a) 1, b) 1, c) 1, d) 1		DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	Faculdade de Ciências U. L. - Dep. Física		
DRAWN	J. Silvestre		09JUL18	TITLE (Description): a) Bottom Front Beam b) Top Front Beam c) Hold 1 d) Hold 2		
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL: Aluminium	Drawing file CSIDA	A4
				WEIGHT: a) 138.76g, b) 138.98g, c) 7.6g, d) 4.85g	SCALE:1:2	SHEET 7 OF 11

a)

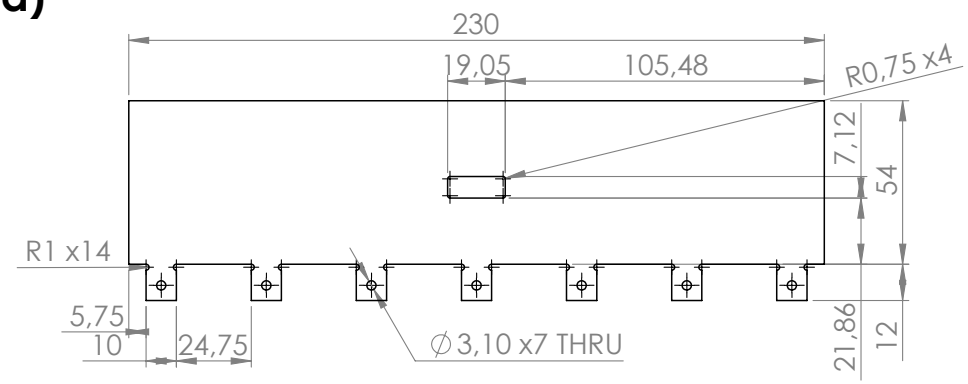


b)

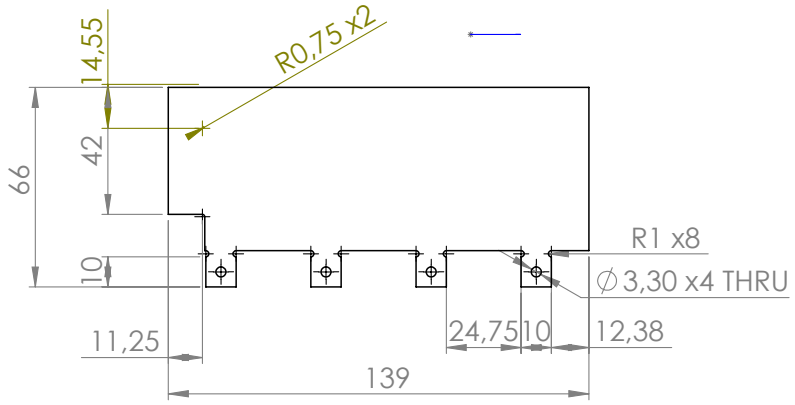


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	Part No.: a) 6, b) 7 FINISH: Quantity: a) 1, b) 1		DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	Faculdade de Ciências U. L. - Dep. Física		
DRAWN	J. Silvestre		09JUL18	TITLE (Description): a) Support for DLP Lightcrafter b) Flange		
CHK'D						
APPV'D						
MFG						
Q.A						
				MATERIAL: Aluminium	Drawing file CSIDA	A4
				WEIGHT: a) 93.19g, b) 108.14g	SCALE:1:2	SHEET 5 OF 11

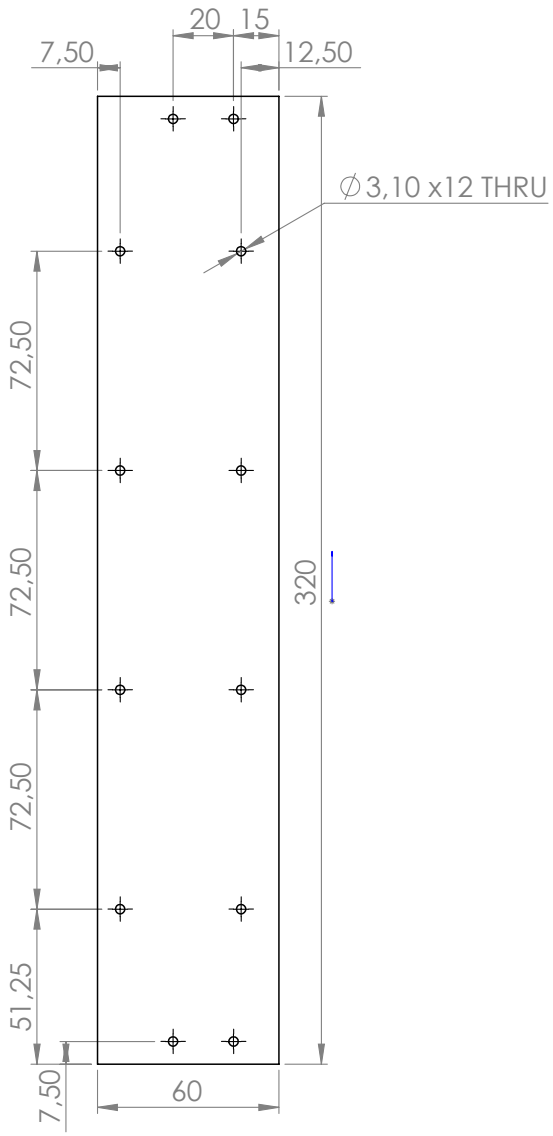
a)



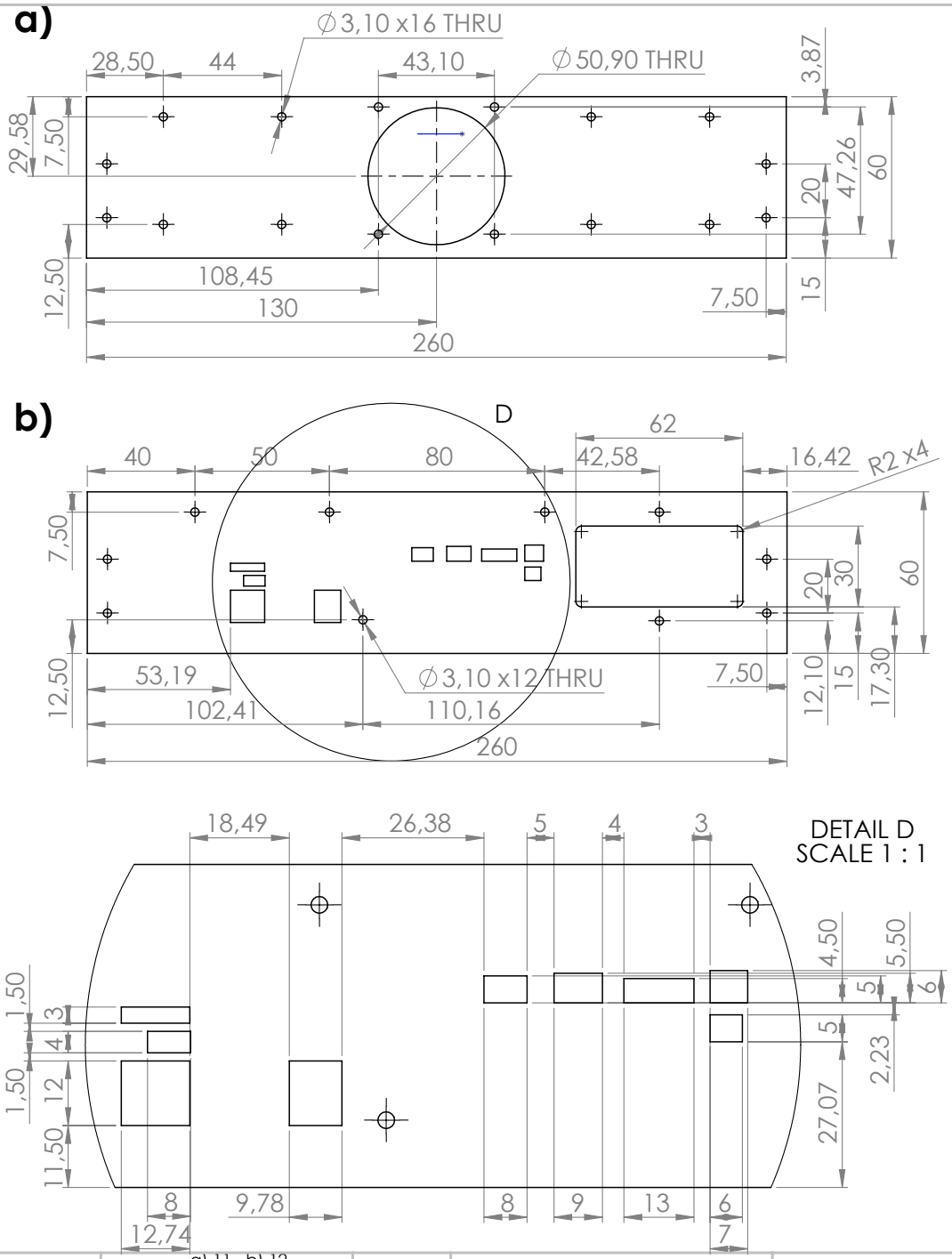
b)

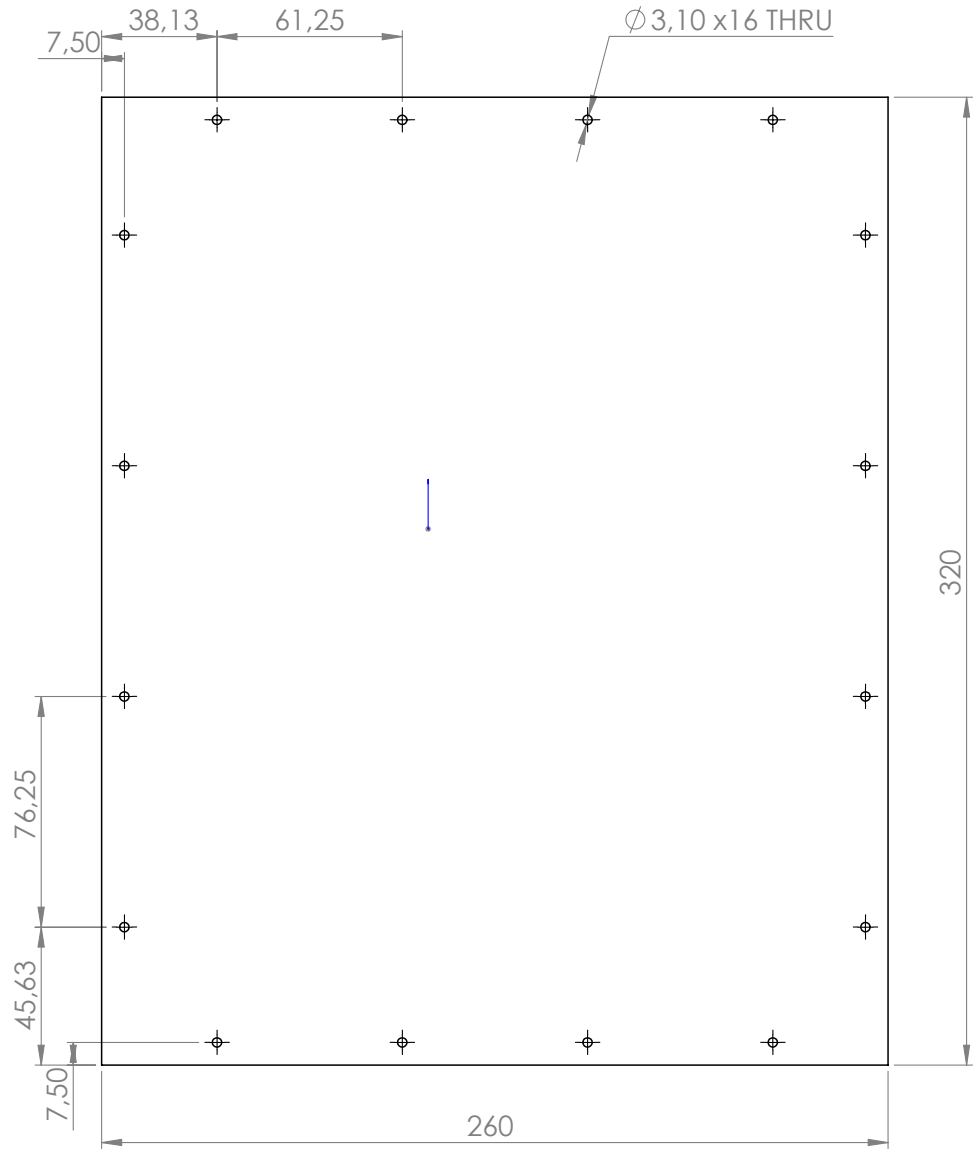


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	Part No.: a) 8, b) 9 FINISH: Quantity: a) 1, b) 2		DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
				Faculdade de Ciências U. L. - Dep. Física		
	NAME	SIGNATURE	DATE	TITLE (Description): Bafflers a) & b)		
DRAWN	J. Silvestre		09JUL18			
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL: 0.5 mm Aluminium sheet	Drawing file CSIDA	A4
				WEIGHT: a) 17.62g, b) 10.42g	SCALE:1:2	SHEET 11 OF 11



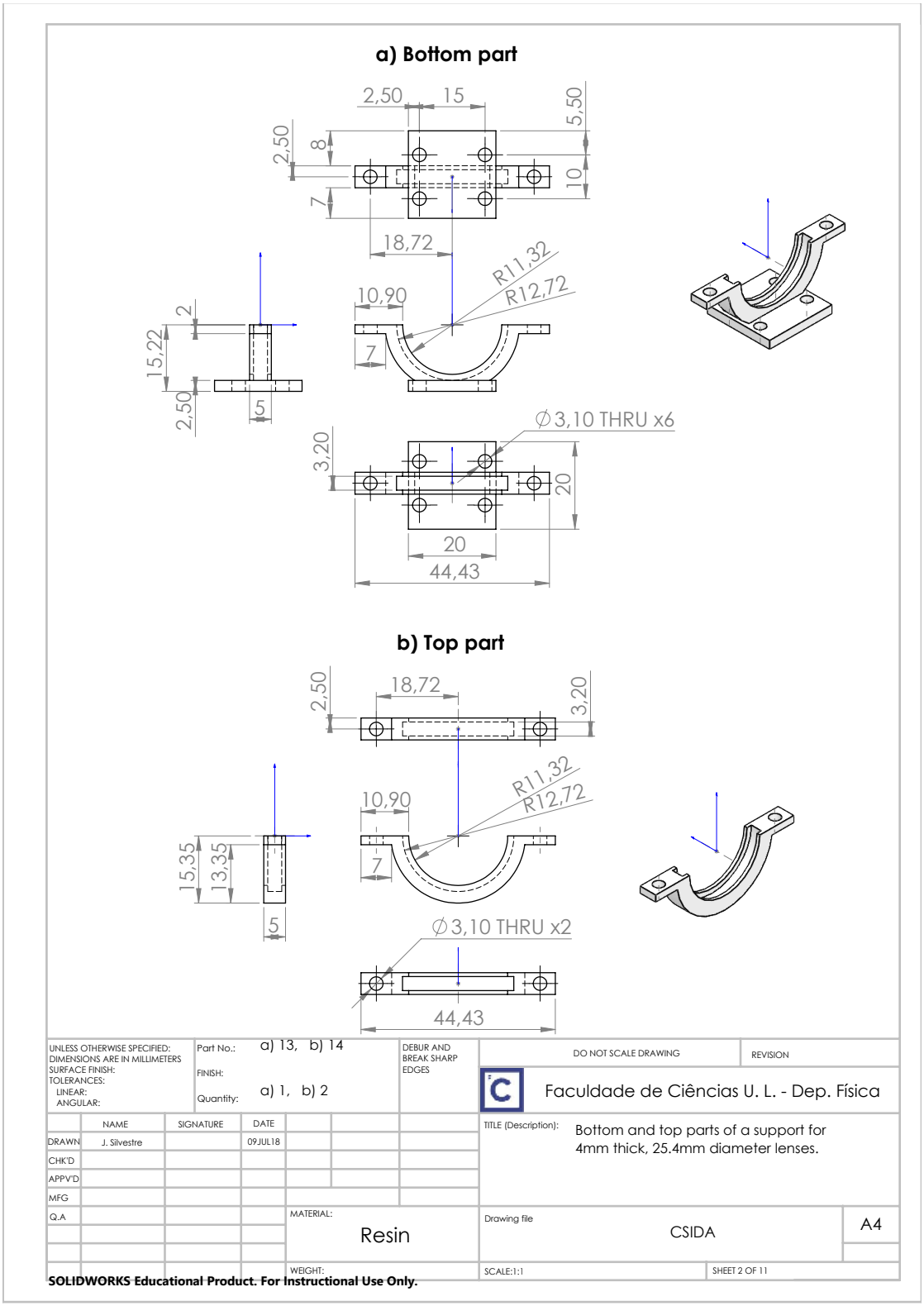
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	Part No.: 10 FINISH: Quantity: 2		DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	Faculdade de Ciências U. L. - Dep. Física		
DRAWN	J. Silvestre		09JUL18	TITLE (Description): Case side sheet		
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL: 0.5 mm Aluminium sheet	Drawing file CSIDA	A4
				WEIGHT: 25.80g	SCALE:1:2	SHEET 9 OF 11


a)
∅3,10 x16 THRU
∅50,90 THRU
28,50
44
43,10
3,87
29,58
7,50
47,26
60
20
12,50
108,45
130
260
7,50
15
b)
D
62
40
50
80
42,58
16,42
R2 x4
7,50
20
30
60
12,50
53,19
102,41
∅3,10 x12 THRU
110,16
260
7,50
12,10
15
17,30
DETAIL D
SCALE 1 : 1
18,49
26,38
5
4
3
4,50
5,50
5
6
1,50
3
4
1,50
12
11,50
5
2,23
27,07
8
9,78
8
9
13
6
12,74
7
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
Part No.: a) 11, b) 12
FINISH:
Quantity: a) 1, b) 1
DEBUR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
Faculdade de Ciências U. L. - Dep. Física
NAME
SIGNATURE
DATE
DRAWN
J. Silvestre
09JUL18
CHK'D
APPV'D
MFG
Q.A
TITLE (Description):
a) Case front sheet
b) Case back sheet
MATERIAL:
0.5 mm Aluminium sheet
WEIGHT: a) 18.15g, b) 17.74g
Drawing file
CSIDA
A4
SCALE:1:2
SHEET 10 OF 11
SOLIDWORKS Educational Product. For Instructional Use Only.

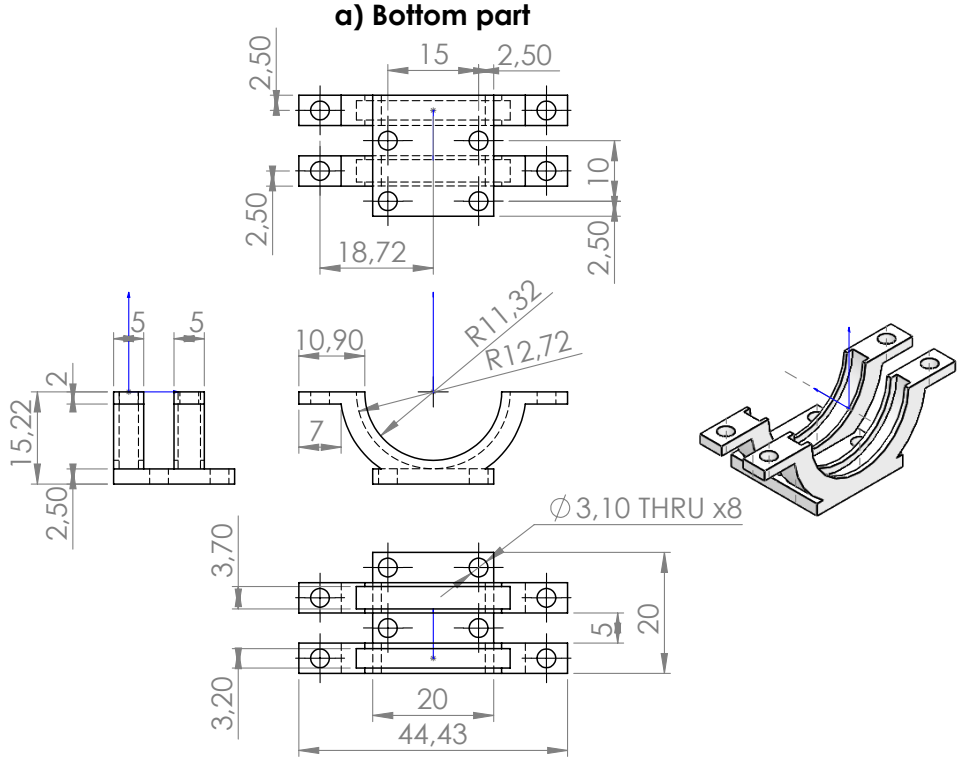


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	Part No.: 20 FINISH: Quantity: 1		DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	Faculdade de Ciências U. L. - Dep. Física		
DRAWN	J. Silvestre		09JUL18	TITLE (Description): Case cover sheet		
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL: 0.5 mm Aluminium sheet	Drawing file CSIDA	A4
				WEIGHT: 112.16g	SCALE:1:3	SHEET 8 OF 11

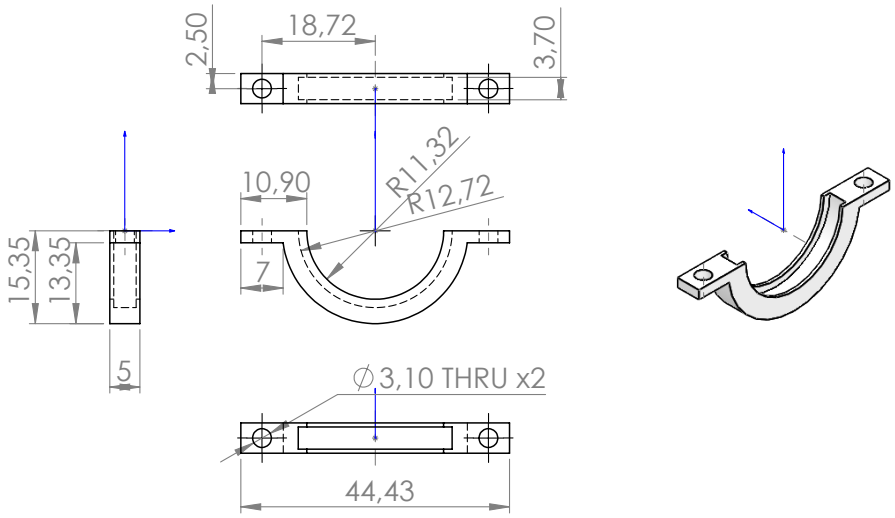
D.2.2 Photopolymer Resin Components



a) Bottom part

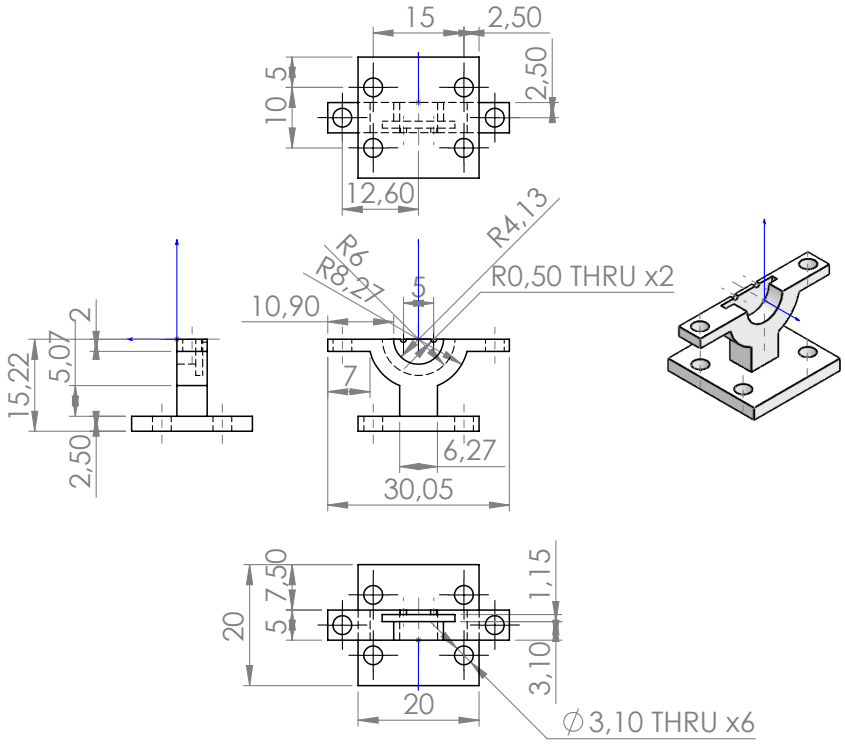


b) Top part

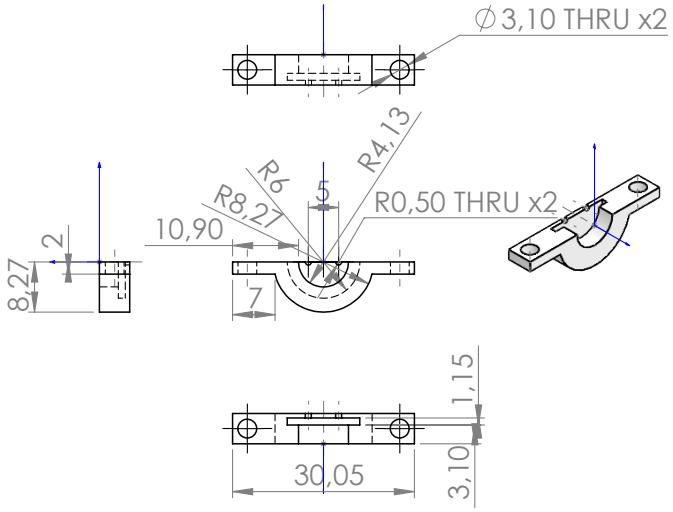


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	Part No.: a) 15, b) 16 FINISH: Quantity: a) 1, b) 1		DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	Faculdade de Ciências U. L. - Dep. Física		
DRAWN	J. Silvestre		09JUL18	TITLE (Description): Bottom part of a support for one 4mm and one 6mm thick, 25.4mm diameter lenses, and top part of a support for 6mm thick, 25.4mm diameter lenses.		
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL: Resin	Drawing file: CSIDA	A4
				WEIGHT:	SCALE:1:1	SHEET 3 OF 11

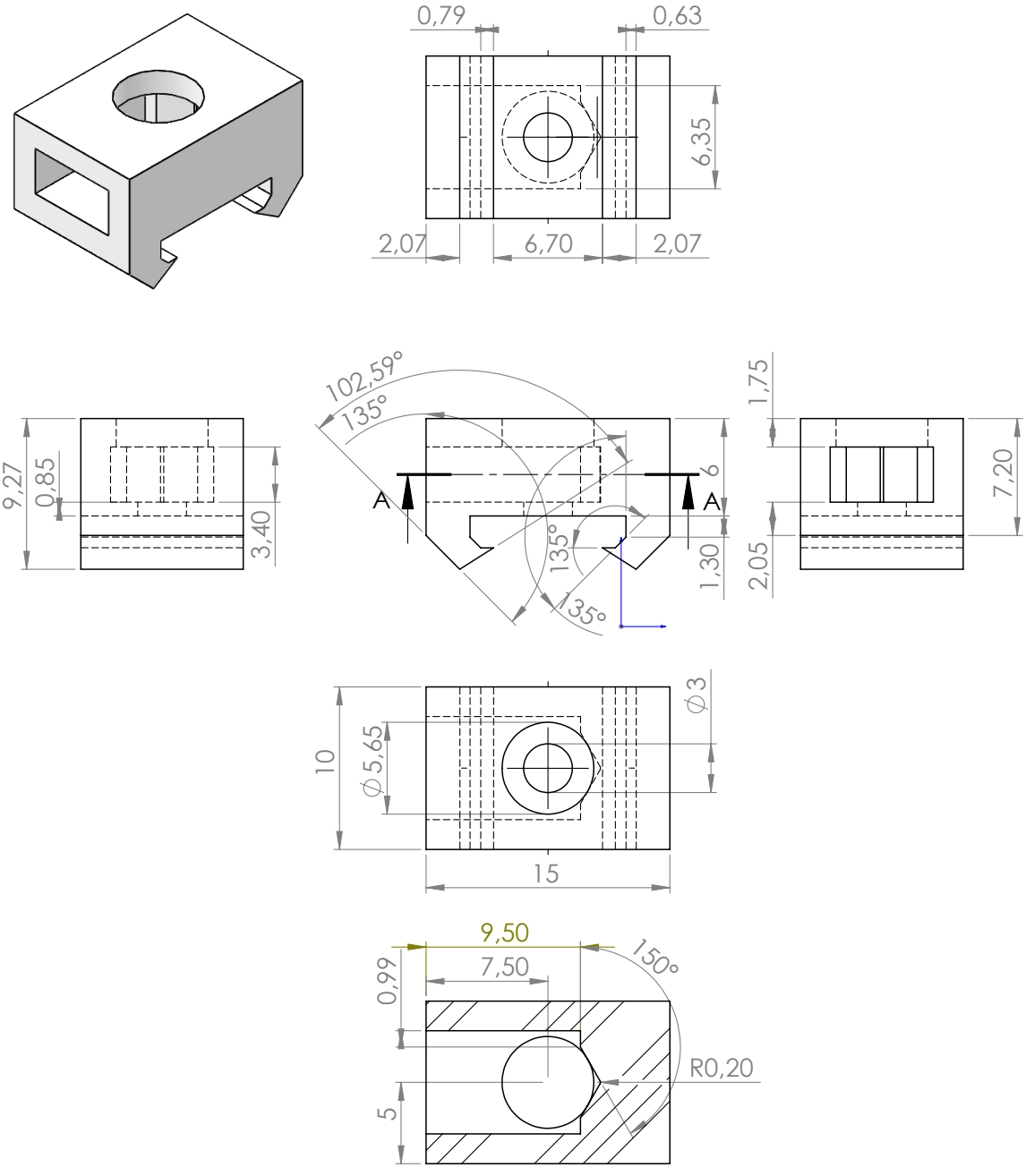
a) Bottom part



b) Top part



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	Part No.: a) 17, b) 18 FINISH: Quantity: a) 1, b) 1		DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
				Faculdade de Ciências U. L. - Dep. Física		
	NAME	SIGNATURE	DATE	TITLE (Description): Bottom and top parts of a support for for Hamamatsu S1223.		
DRAWN	J. Silvestre		09JUL18			
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL: Resin	Drawing file CSIDA	A4
				WEIGHT:	SCALE:1:1	SHEET 4 OF 11



SECTION A-A

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	Part No.: 19 FINISH: Quantity: 6		DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	Faculdade de Ciências U. L. - Dep. Física		
DRAWN	J. Silvestre		09JUL18	TITLE (Description): Bearing brake		
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL: Resin	Drawing file CSIDA	A4
				WEIGHT:	SCALE:3:1	SHEET 1 OF 11

D.3 List of Bought Parts

- 1× Lens CSOPTLENS01 - 111-0210E
- 2× Lens CSOPTLENS01 - 111-0222E
- 8× M2 Hexagonal Spacer, 5 mm M/F
- 12× M2.5 Screw Pozidriv, 6 mm
- 8× M2.5 Nut
- 4× M3 Hexagonal Spacer, 5 mm Female/10 mm Male
- 4× M3 Hexagonal Spacer, 6 mm M/F
- 10× M3 Screw Pozidriv, 5 mm
- 8× M3 Screw Slot, 10 mm
- 39× M3 Screw Slot, 12 mm
- 8× M3 Screw Slot, 20 mm
- 103× M3 Self-locking Nut
- 66× M3 Square headed Bolt, 6 mm
- 18× M3 Square headed Bolt, 12 mm
- 51× M3 Nuts
- 3× MakerBeam Linear Slide Carriage
- 1× MakerBeam Linear Slide Rail, 300 mm (from which an 80 mm and a 140 mm rails were produced)
- 4× OpenBeam, 45 mm (from which four 40 mm beams were produced)
- 5× OpenBeam, 270 mm (one of which was used to make a 230 mm beam)
- 4× OpenBeam Corner Cubes, 15×15×15 mm

D.4 Exploded View

Here we present an exploded view of the structural subsystem, while also showing the relative placement of the optical subsystem, LightCrafter and Arduino.

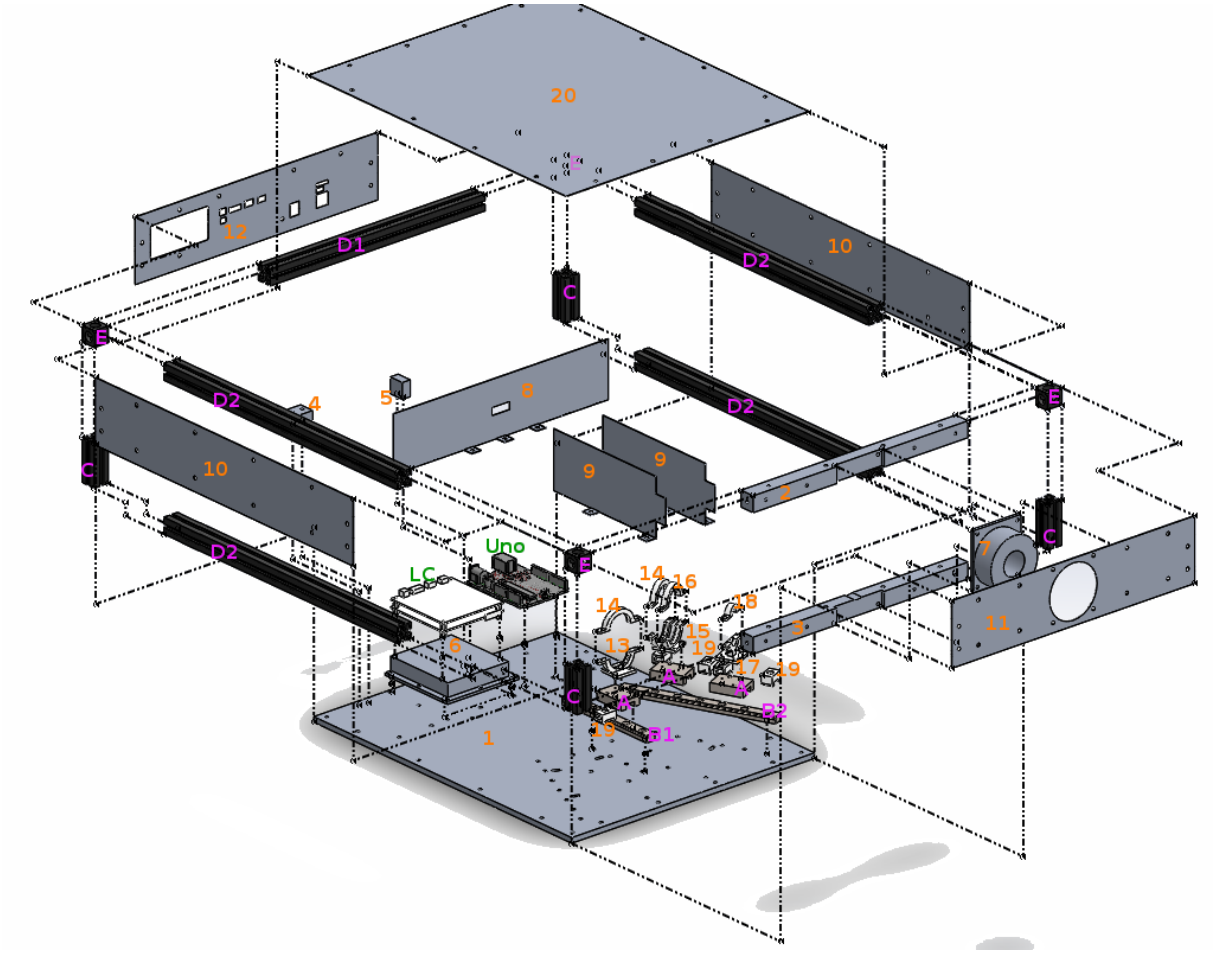


Figure D.1: *Exploded view of the structural subsystem.* ***A)*** *slide carriage;* ***B1)*** *slide rail - 80 mm;* ***B2)*** *slide rail - 140 mm;* ***C)*** *OpenBeam - 40 mm;* ***D1)*** *OpenBeam - 230 mm;* ***D2)*** *OpenBeam - 270 mm;* ***E)*** *Corner cubes;* ***LC)*** *DLP LightCrafter 3000;* ***UNO)*** *Arduino Uno;* ***1)*** *Base;* ***2)*** *Front bottom beam;* ***3)*** *Front top beam;* ***4)*** *Hold 1;* ***5)*** *Hold 2;* ***6)*** *Support for DLP LightCrafter 3000;* ***7)*** *Flange;* ***8)*** *Electronics baffle;* ***9)*** *Signal path baffle;* ***10)*** *Case side panel;* ***11)*** *Case front panel;* ***12)*** *Case back panel;* ***13)*** *Support for a ∅25,4 mm, 4 mm thick, lens - bottom part;* ***14)*** *Support for a ∅25,4 mm, 4 mm thick, lens - top part;* ***15)*** *Support for two ∅25,4 mm, 4 mm and 6 mm thick, lenses - bottom part;* ***16)*** *Support for a ∅25,4 mm, 6 mm thick, lens - top part;* ***17)*** *Hamamatsu S1223 support - bottom part ;* ***18)*** *Hamamatsu S1223 support - top part;* ***19)****Bearing brake ;* ***20)*** *Case cover;*

D.5 Simulations

In this appendix we present the results of simulations performed to evaluate the adequacy of the design for the structural subsystem.

D.5.1 Mesh

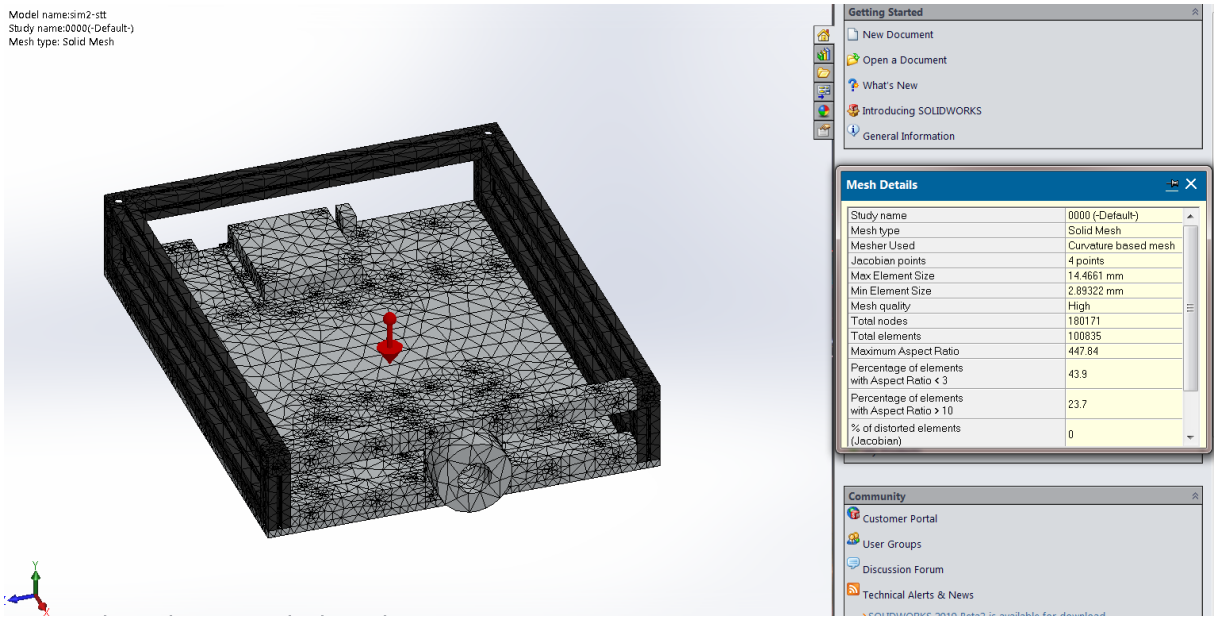


Figure D.2: *Visualization of the mesh that was applied in order to run the FEM simulations.*

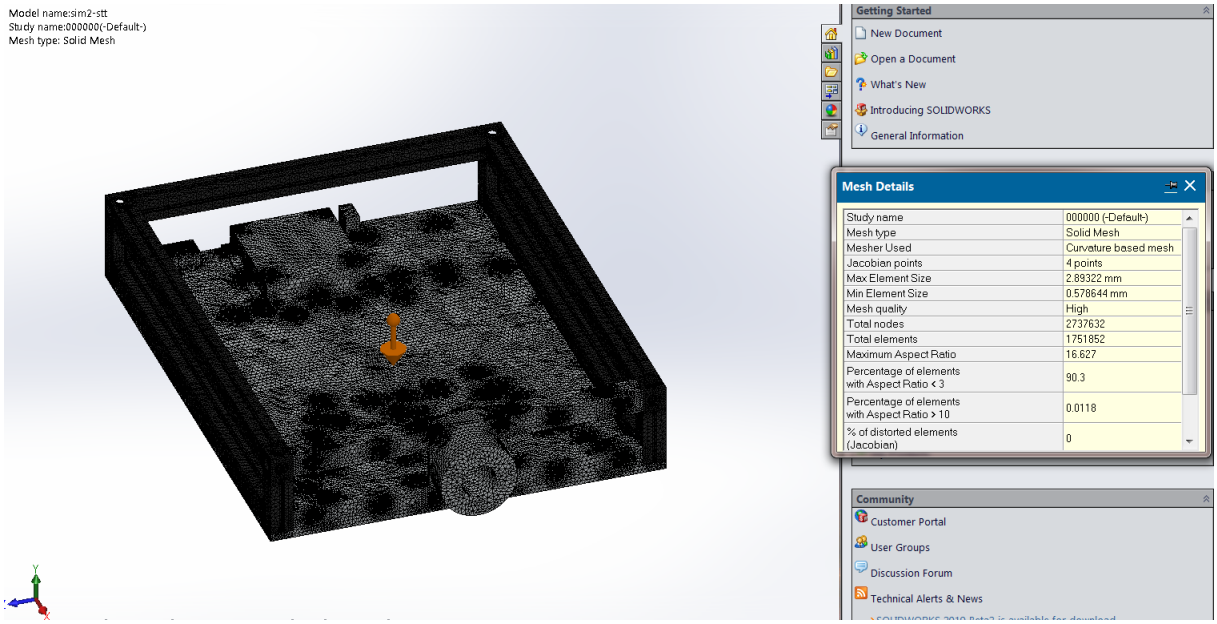


Figure D.3: *Visualization of a finner mesh that was applied in order to run a few control simulations.*

D.5.2 Results for T = 0 °C

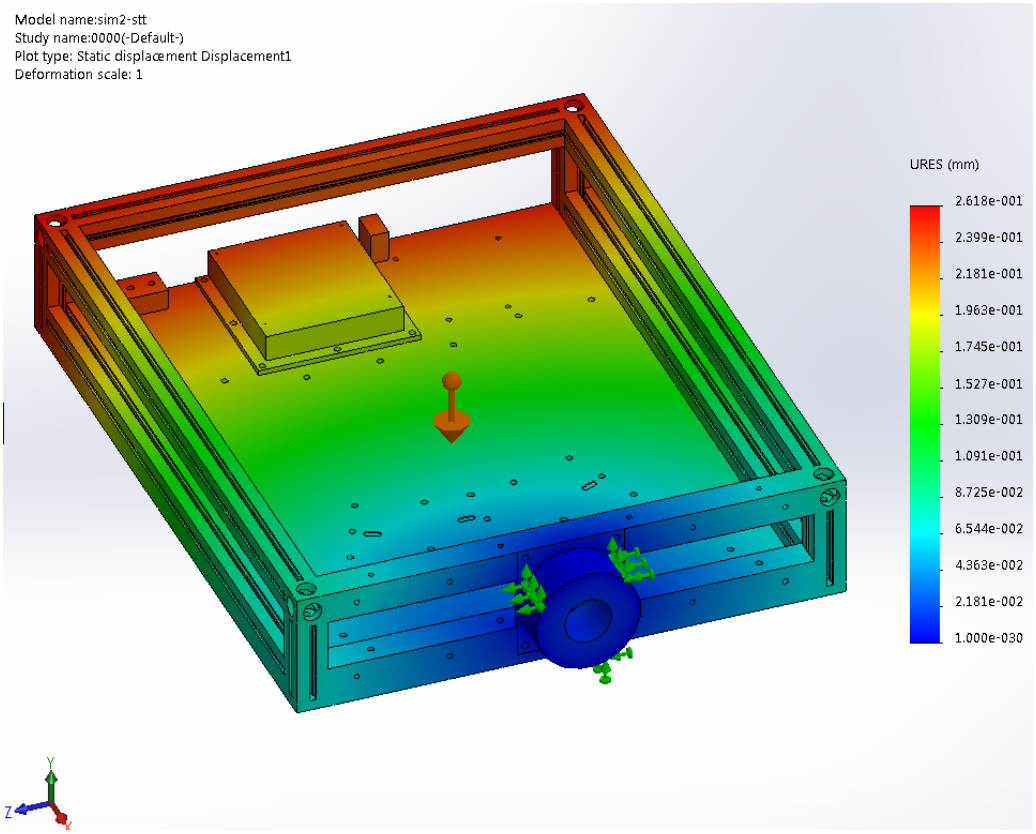


Figure D.4: *(δ,ϵ,ζ)=(90°,0°,90°).*

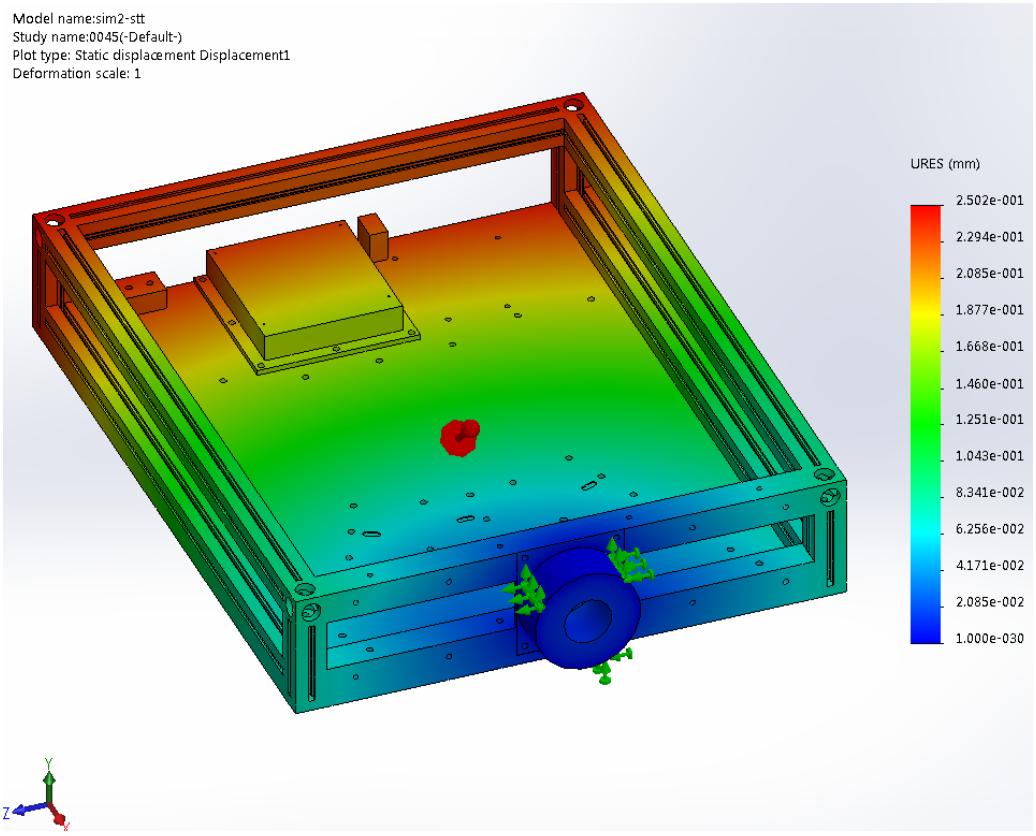


Figure D.5: *(δ,ϵ,ζ)=(45°,45°,90°).*

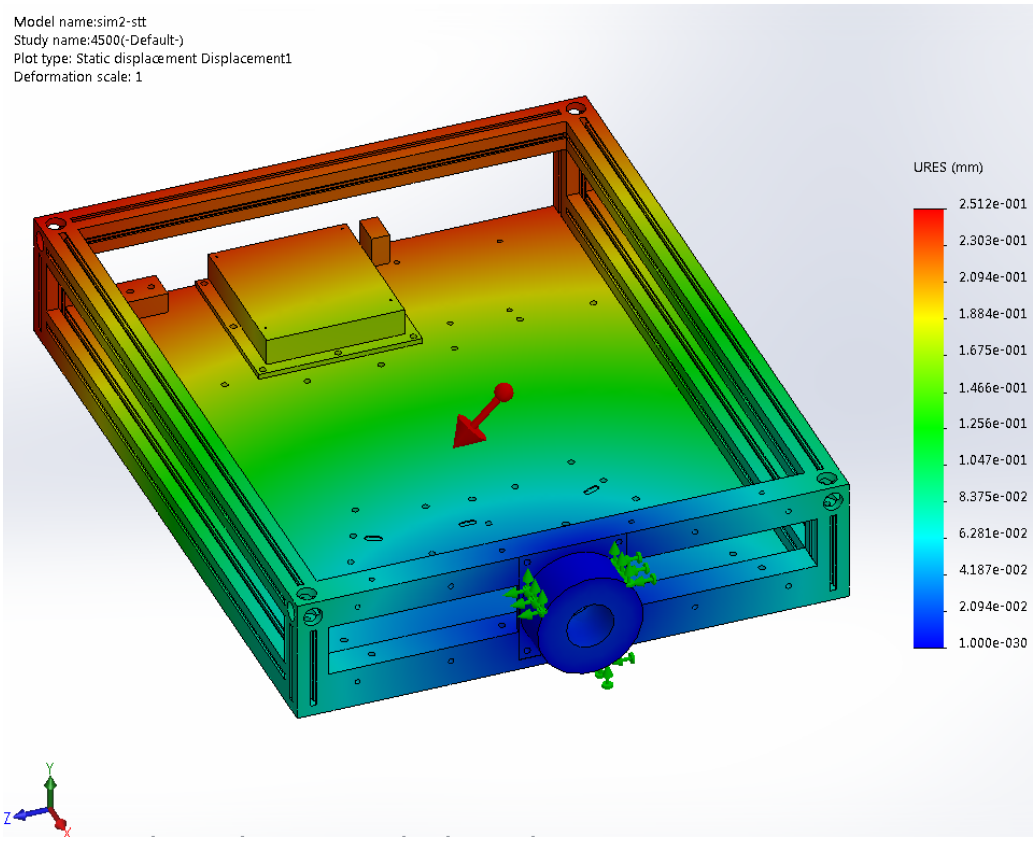


Figure D.6: *(δ,ϵ,ζ)=(90°,45°,45°).*

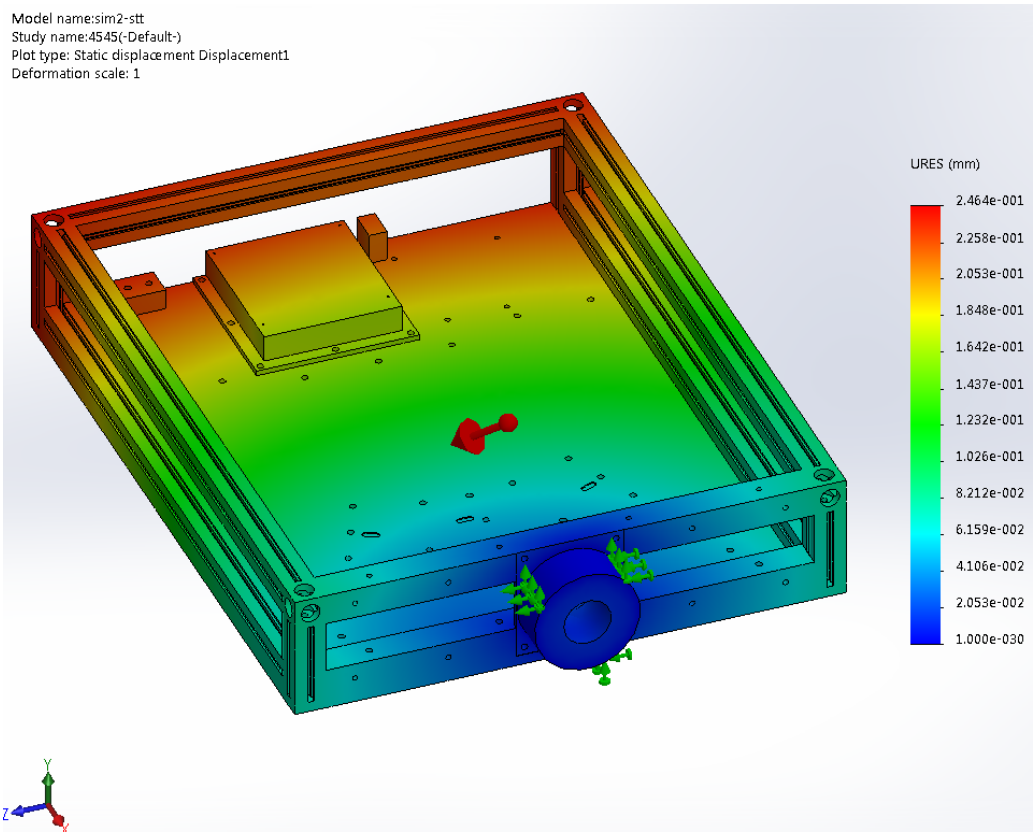


Figure D.7: *(δ,ϵ,ζ)=(45°,45°,45°).*

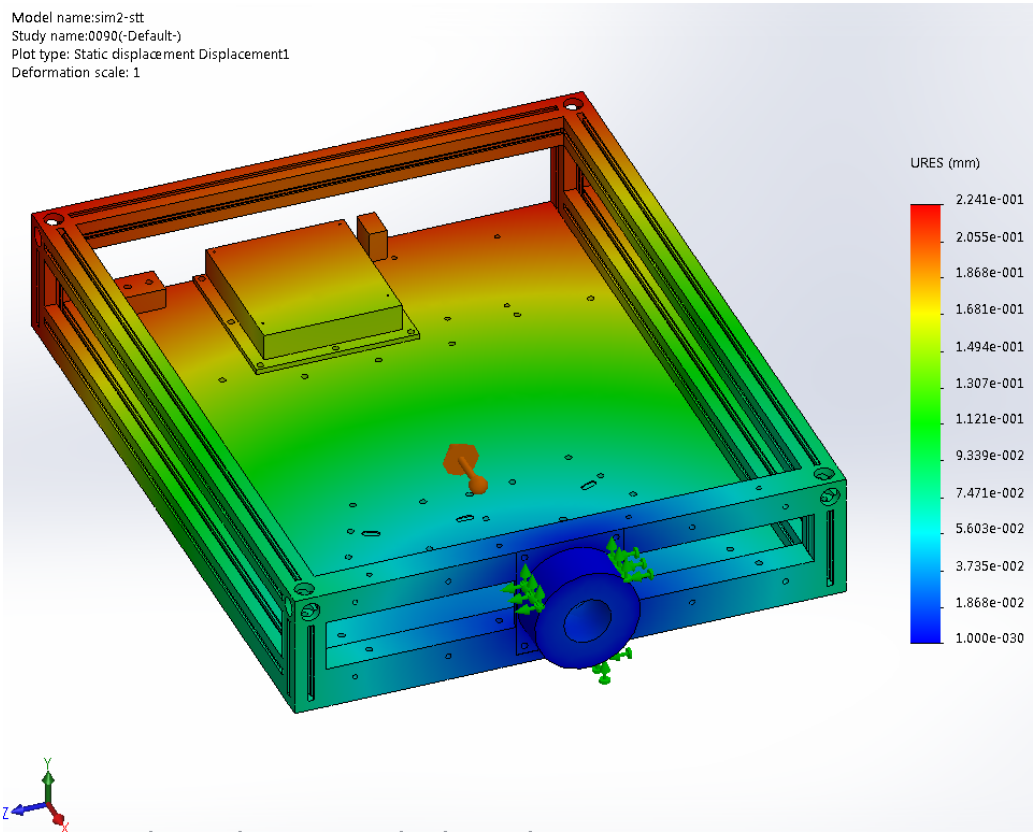


Figure D.8: *(δ,ϵ,ζ)=(0°,90°,90°).*

D.5.3 Results for T = 10 °C

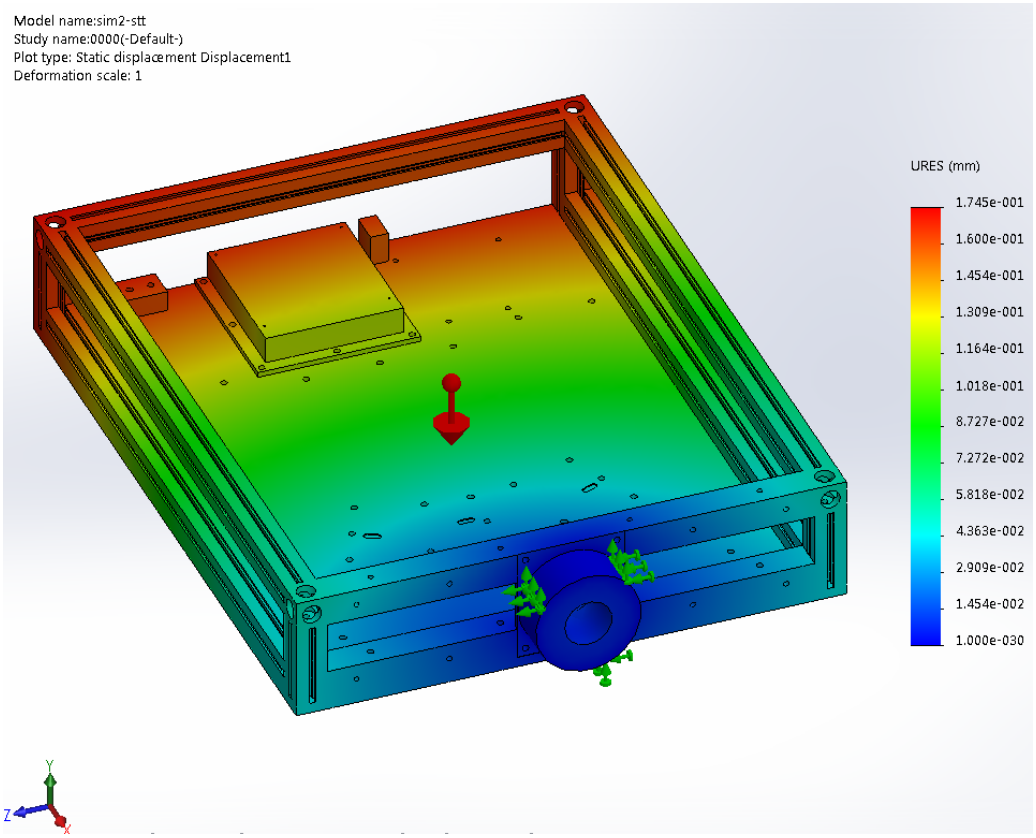


Figure D.9: *(δ,ϵ,ζ)=(90°,0°,90°).*

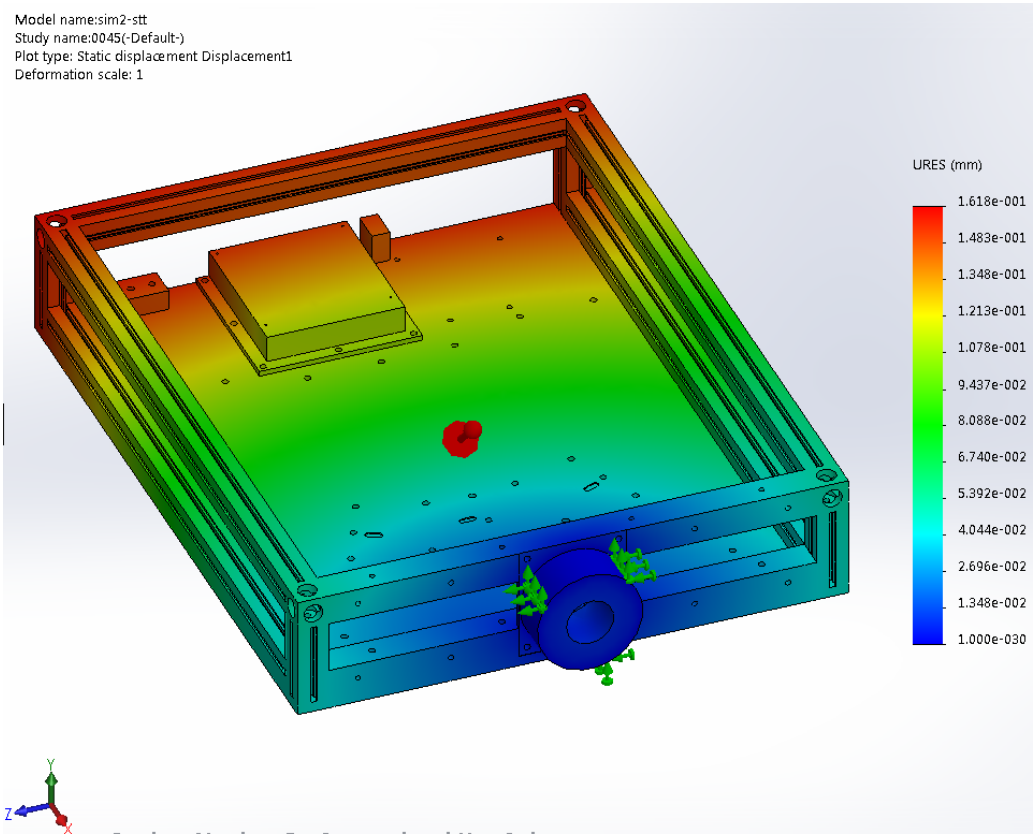


Figure D.10: *(δ,ϵ,ζ)=(45°,45°,90°).*

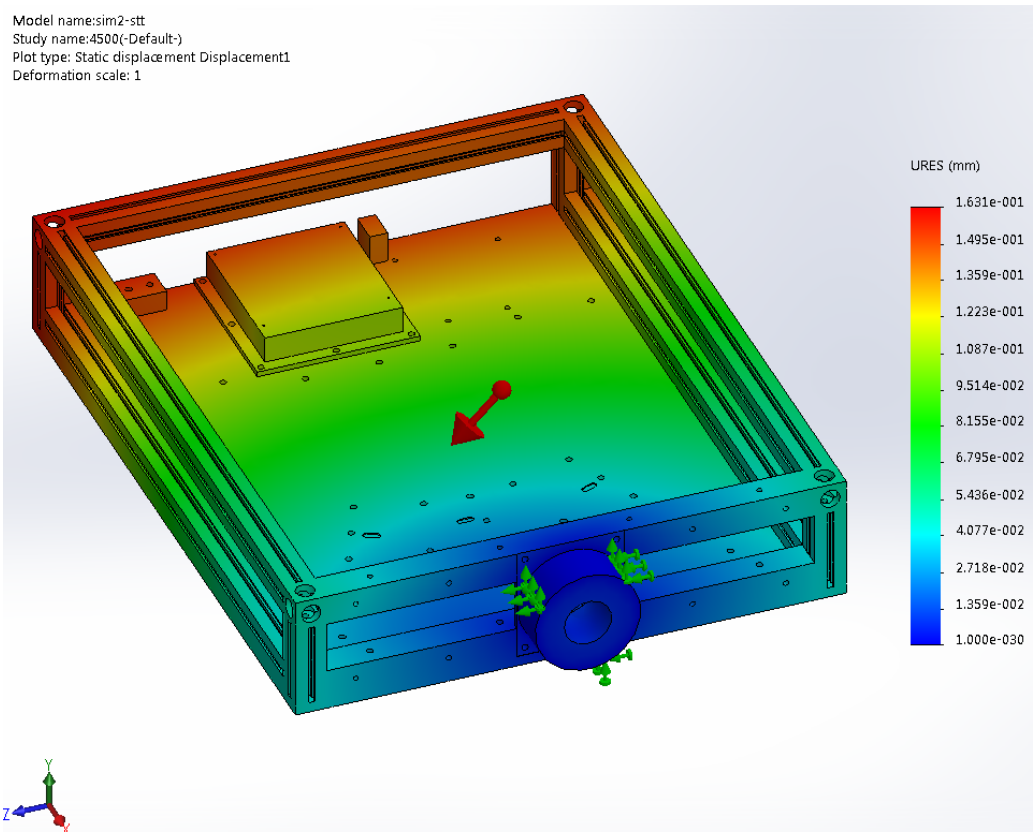


Figure D.11: *(δ,ϵ,ζ)=(90°,45°,45°).*

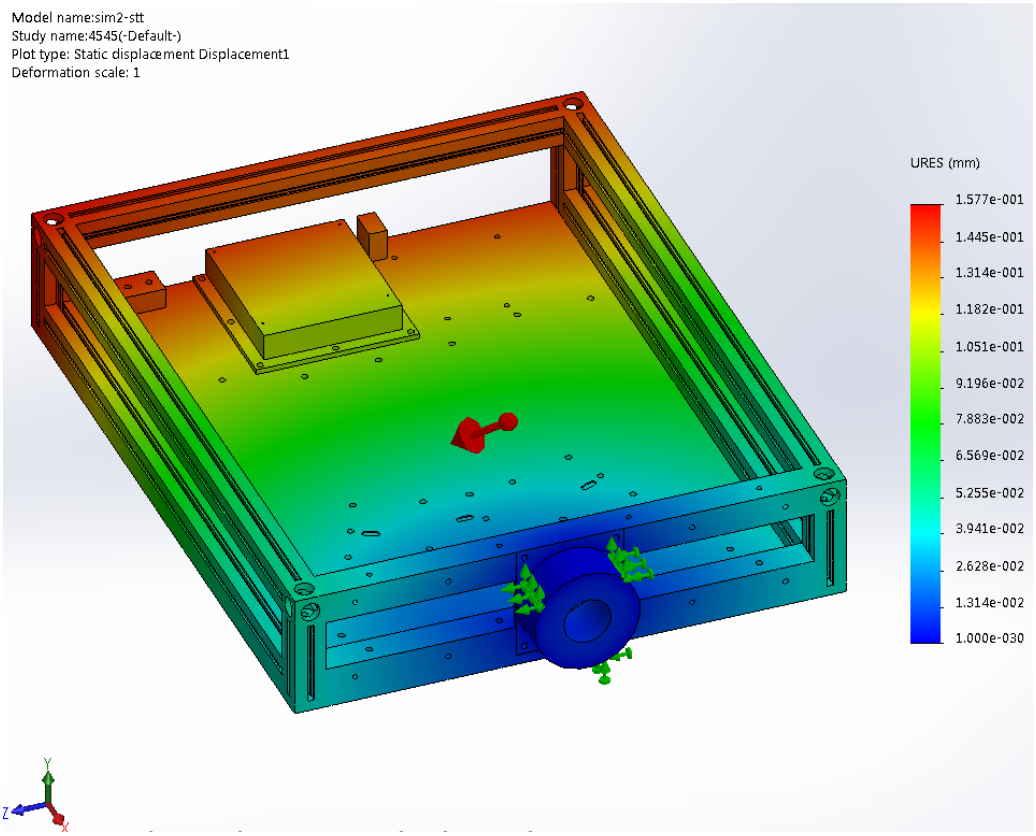


Figure D.12: *(δ,ϵ,ζ)=(45°,45°,45°).*

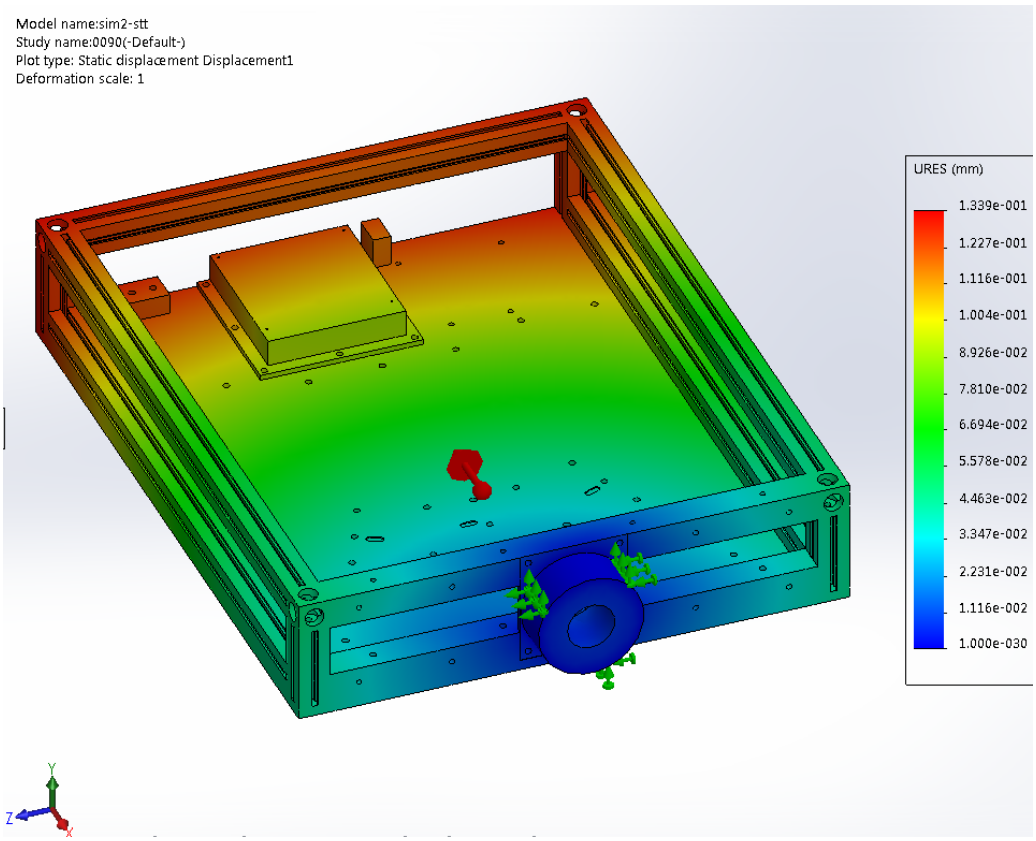


Figure D.13: *(δ,ϵ,ζ)=(0°,90°,90°).*

D.5.4 Results for T = 20 °C

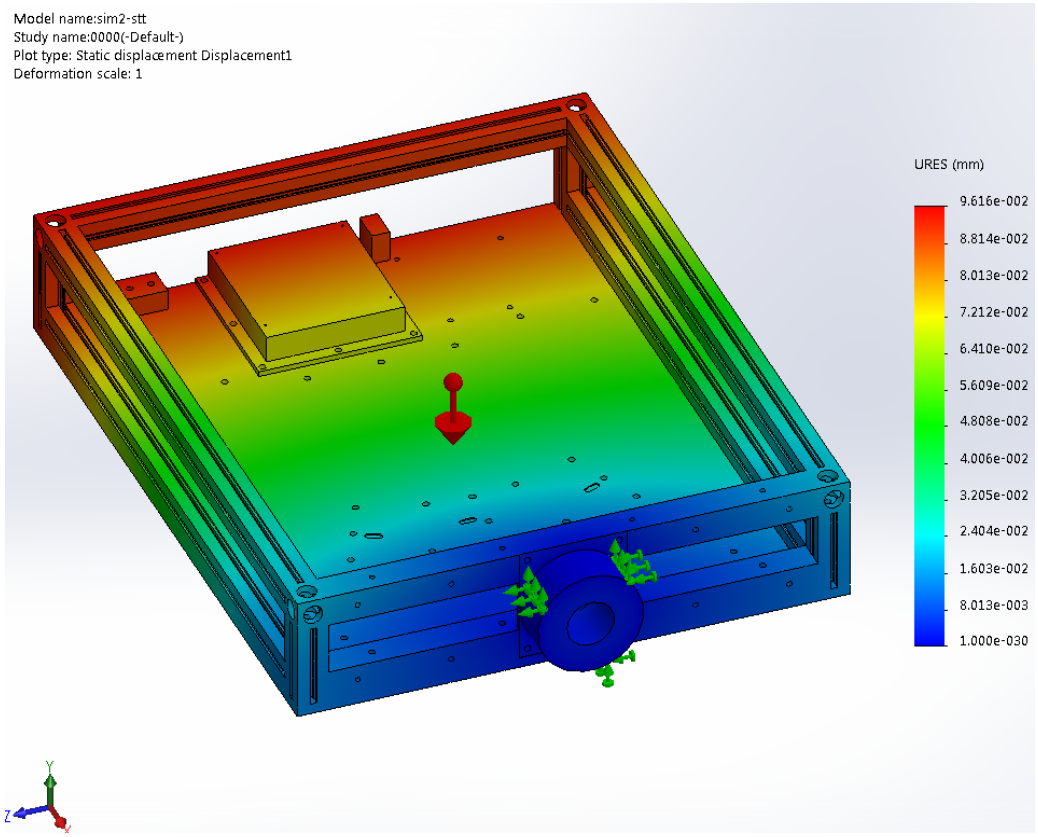


Figure D.14: *(δ,ϵ,ζ)=(90°,0°,90°).*

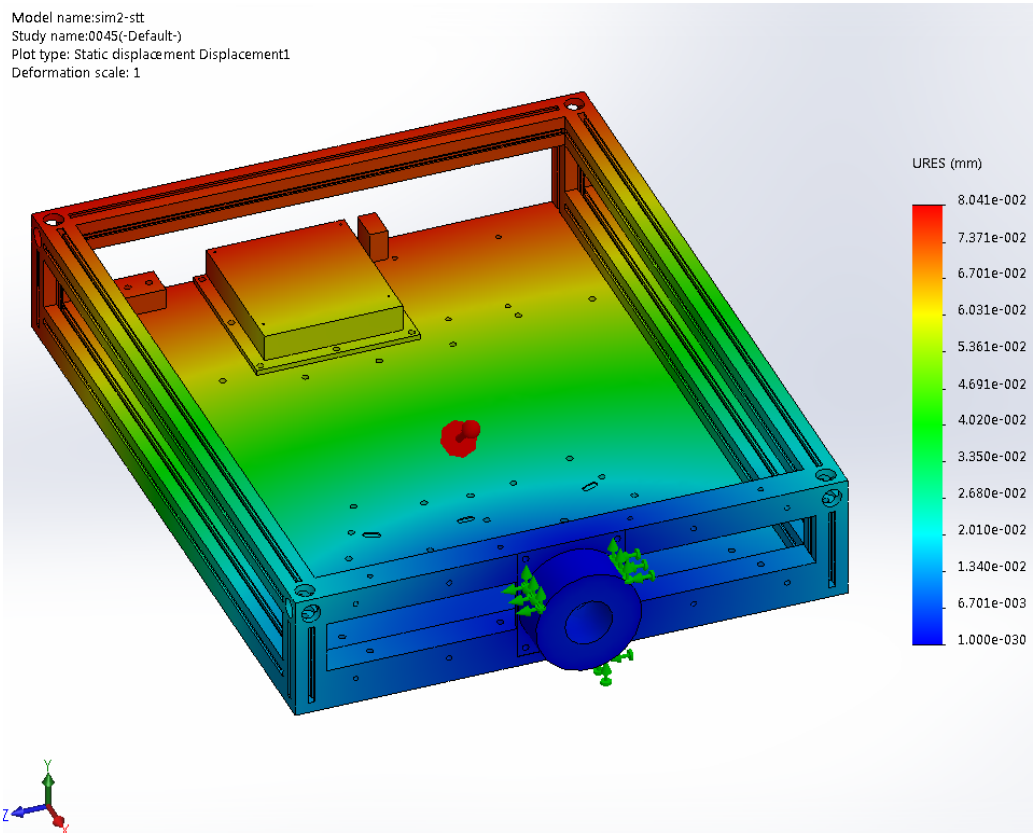


Figure D.15: *(δ,ϵ,ζ)=(45°,45°,90°).*

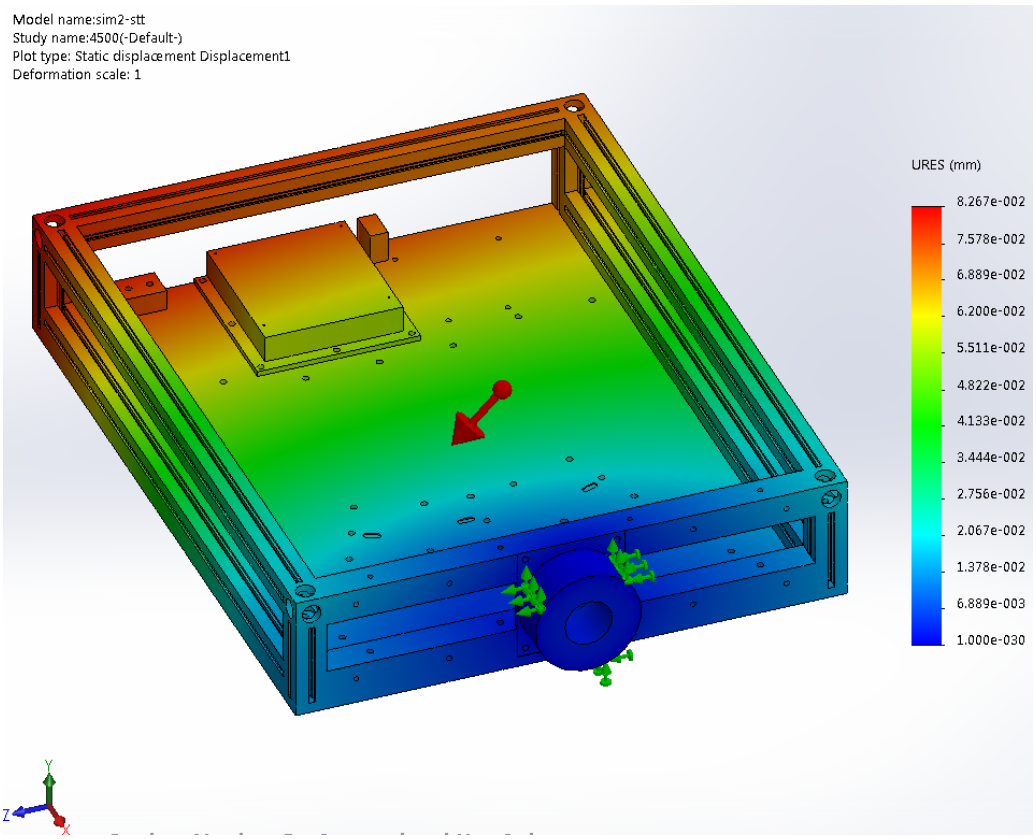


Figure D.16: *(δ,ϵ,ζ)=(90°,45°,45°).*

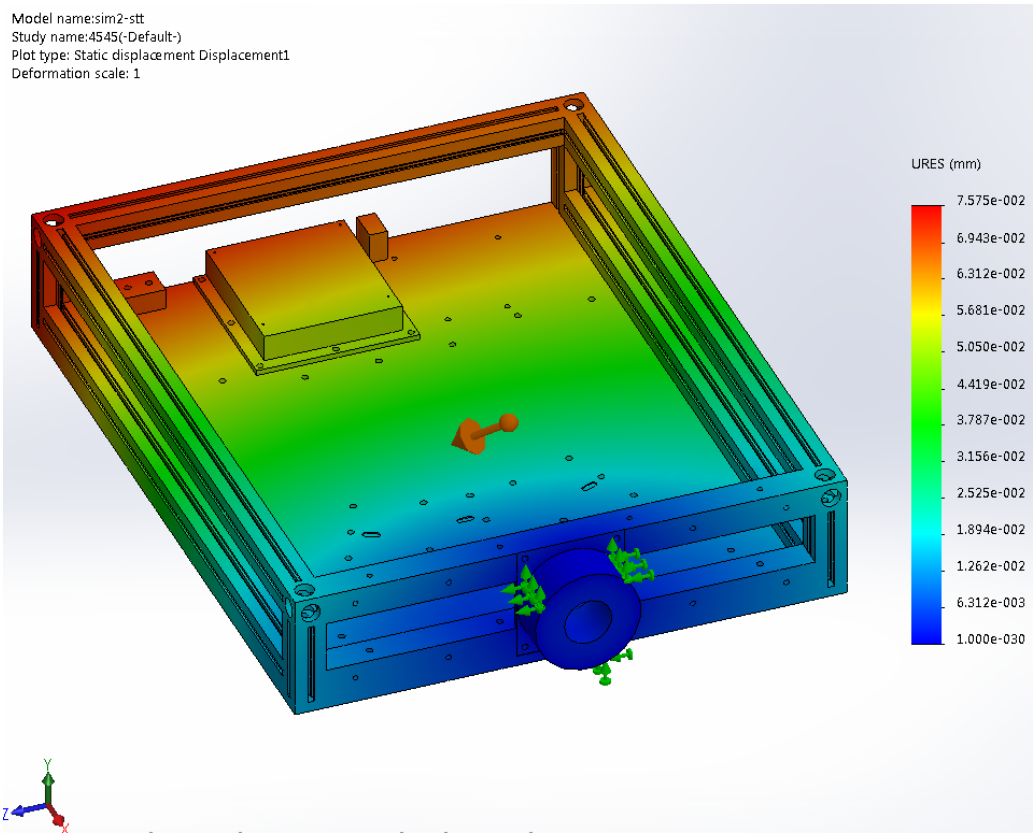


Figure D.17: *(δ,ϵ,ζ)=(45°,45°,45°).*

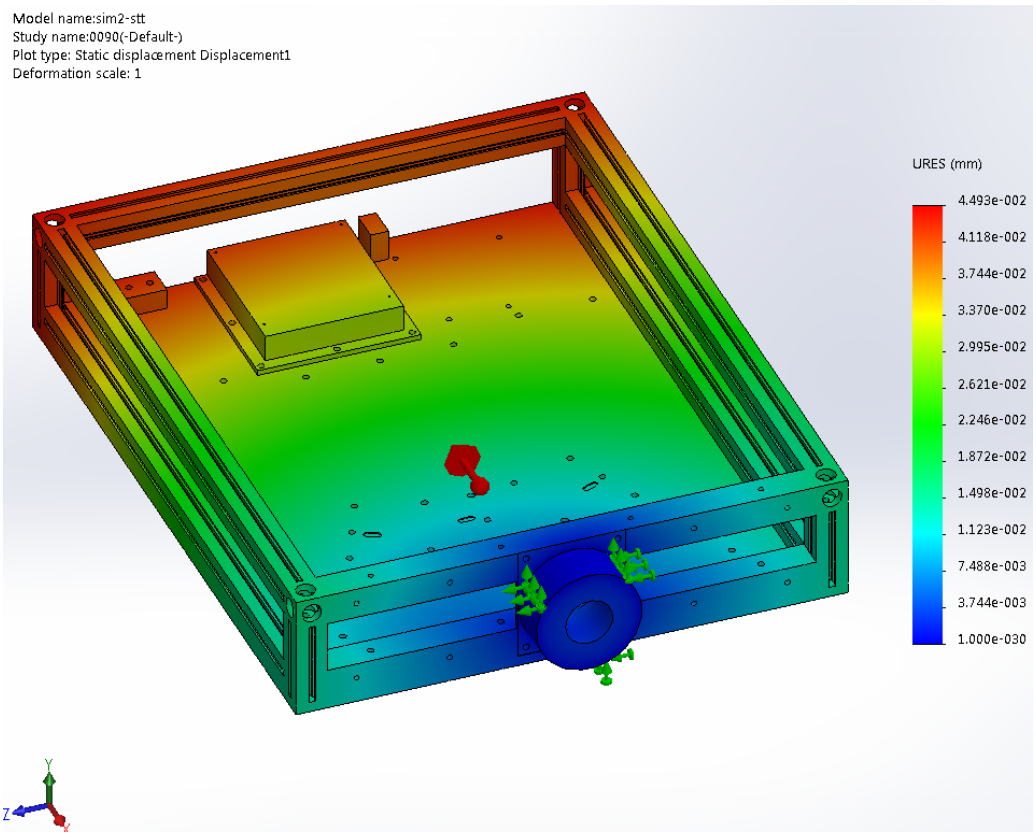


Figure D.18: *(δ,ϵ,ζ)=(0°,90°,90°).*

D.5.5 Results for T = 30 °C

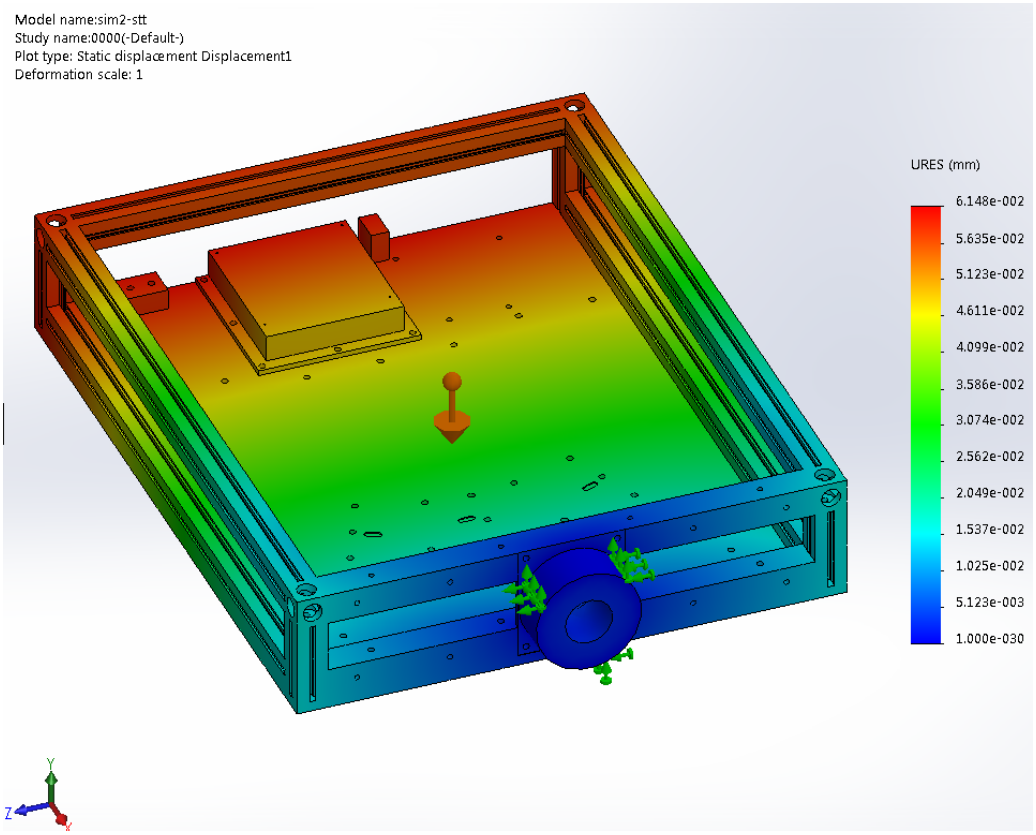


Figure D.19: *(δ,ϵ,ζ)=(90°,0°,90°).*

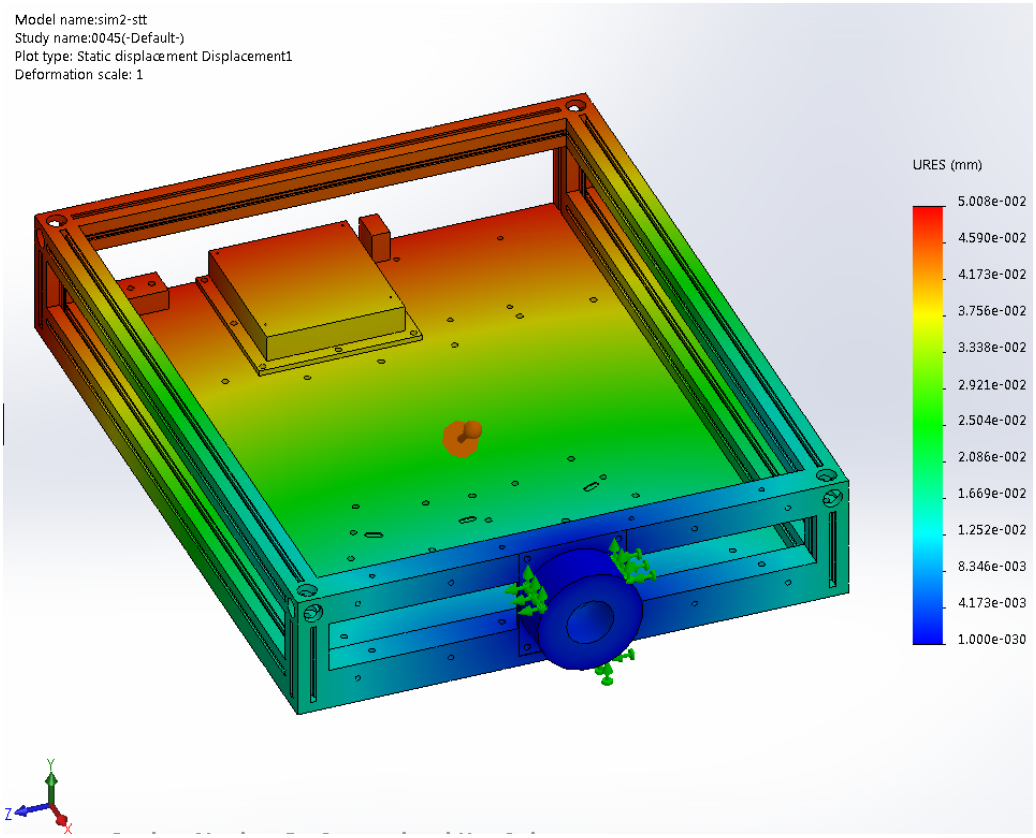


Figure D.20: *(δ,ϵ,ζ)=(45°,45°,90°).*

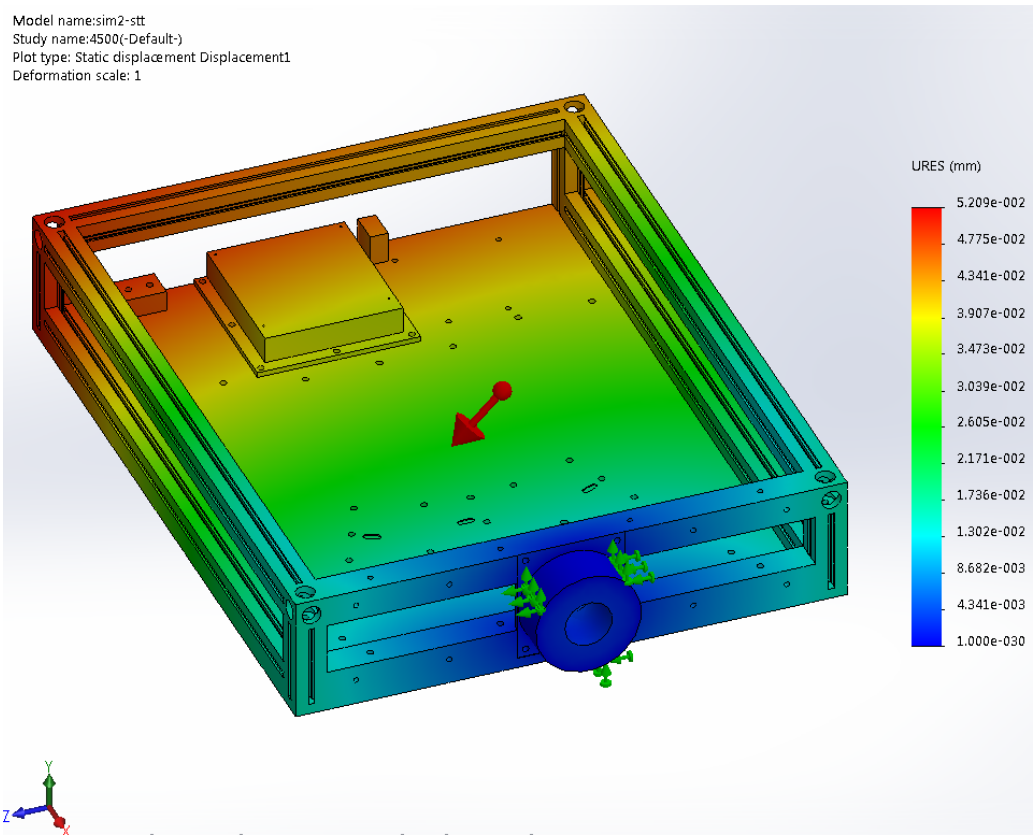


Figure D.21: *(δ,ϵ,ζ)=(90°,45°,45°).*

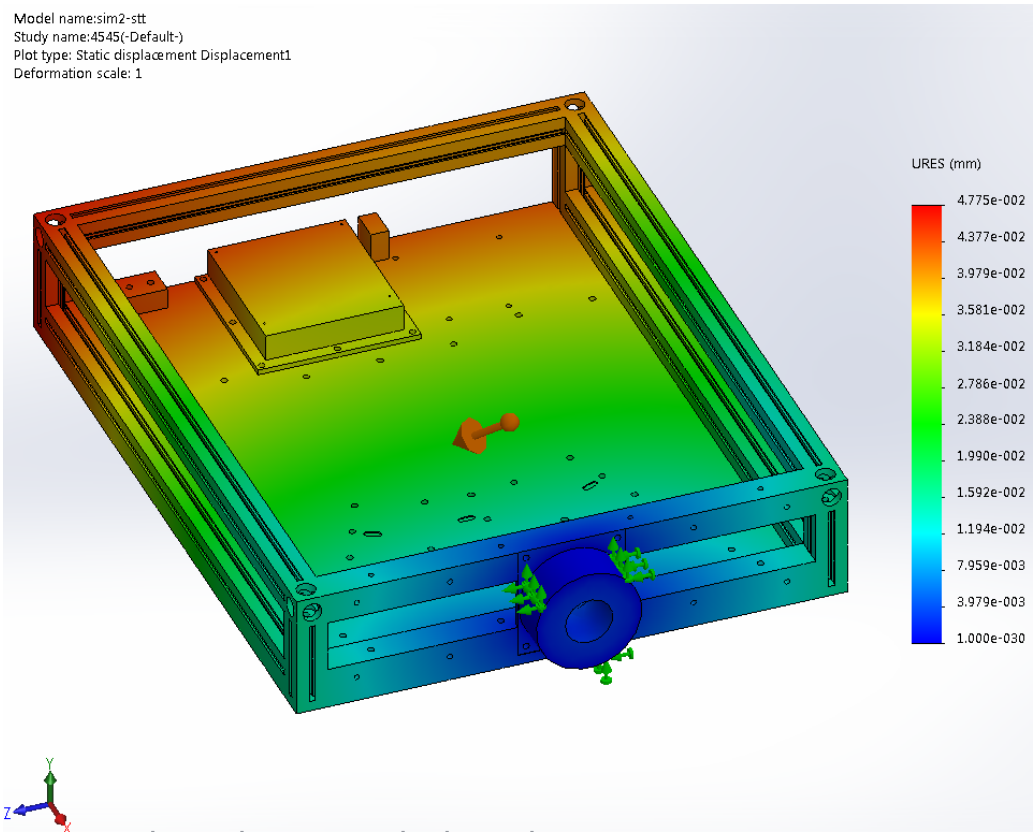


Figure D.22: *(δ,ϵ,ζ)=(45°,45°,45°).*

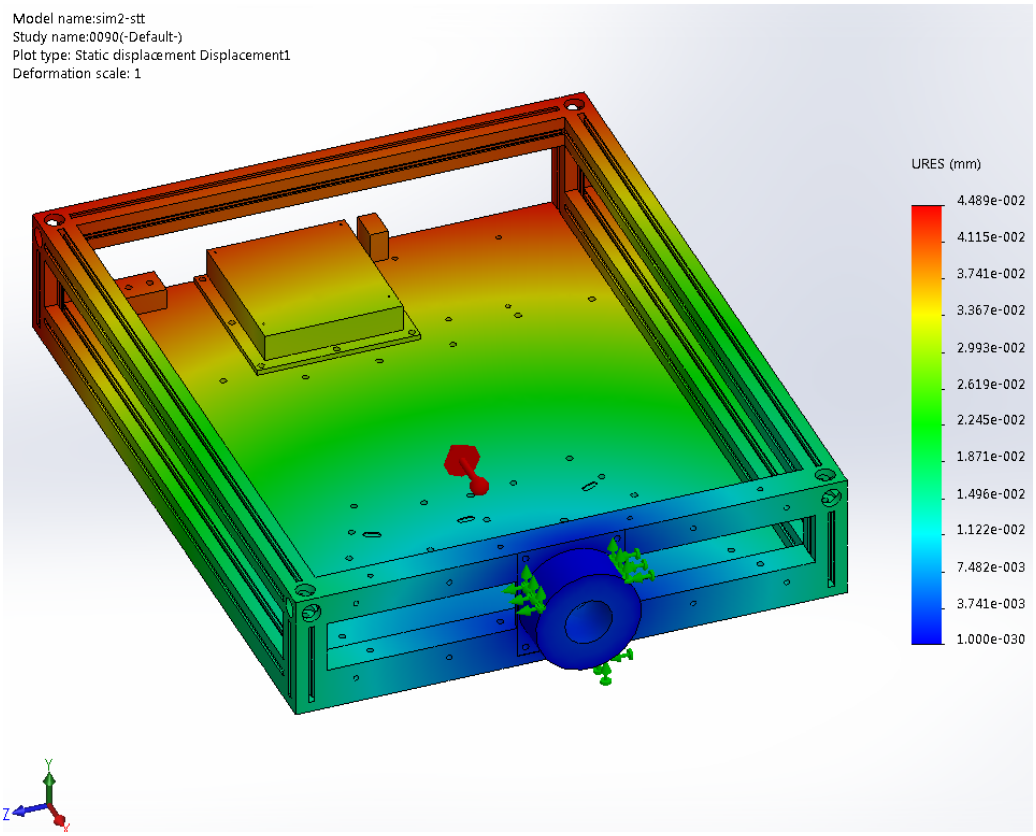


Figure D.23: *(δ,ϵ,ζ)=(0°,90°,90°).*

D.5.6 Results for T = 40 °C

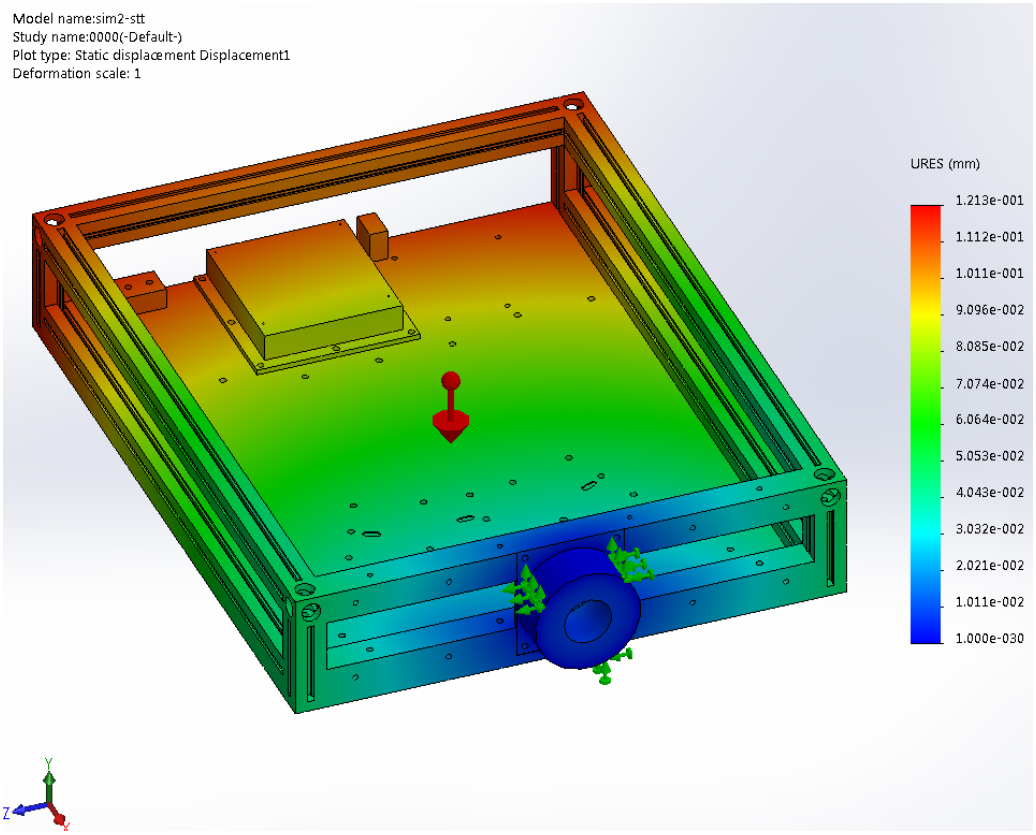


Figure D.24: *(δ,ϵ,ζ)=(90°,0°,90°).*

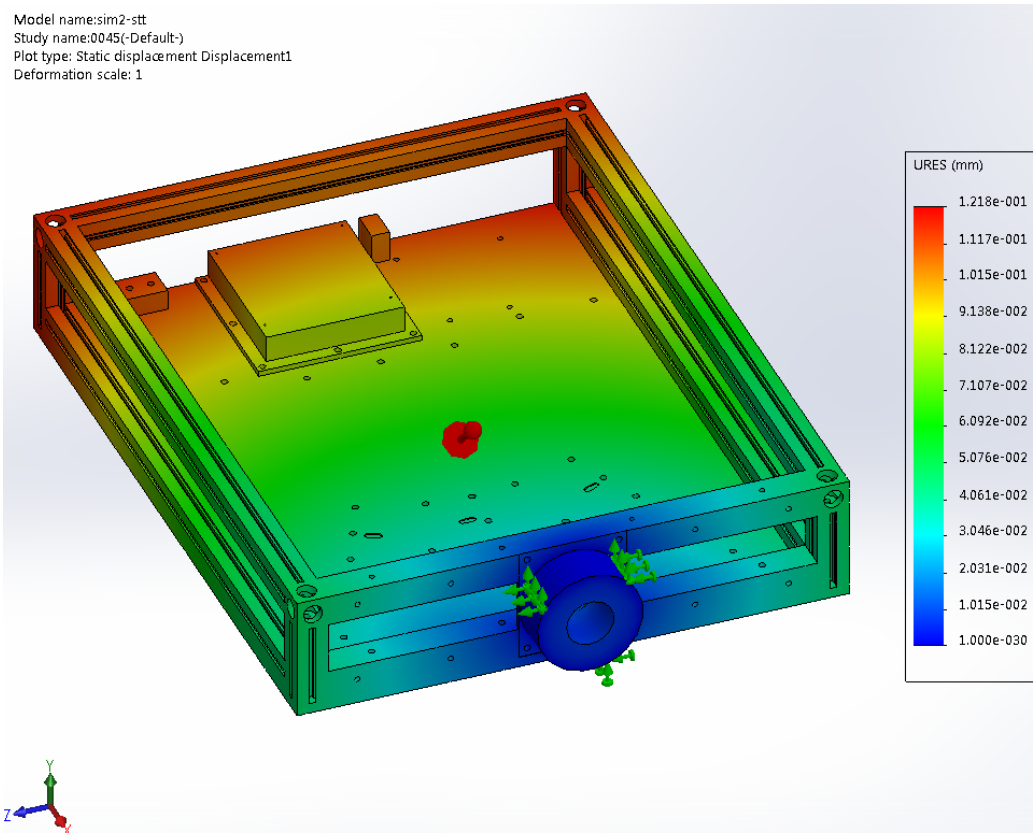


Figure D.25: *(δ,ϵ,ζ)=(45°,45°,90°).*

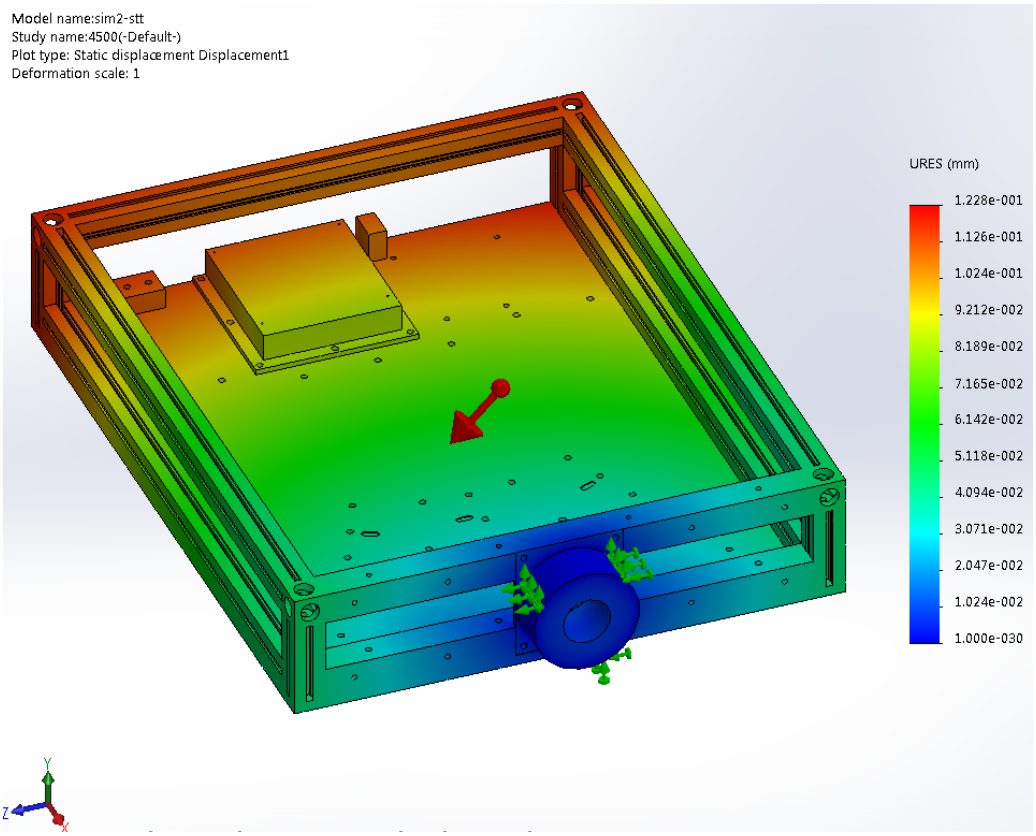


Figure D.26: *(δ,ϵ,ζ)=(90°,45°,45°).*

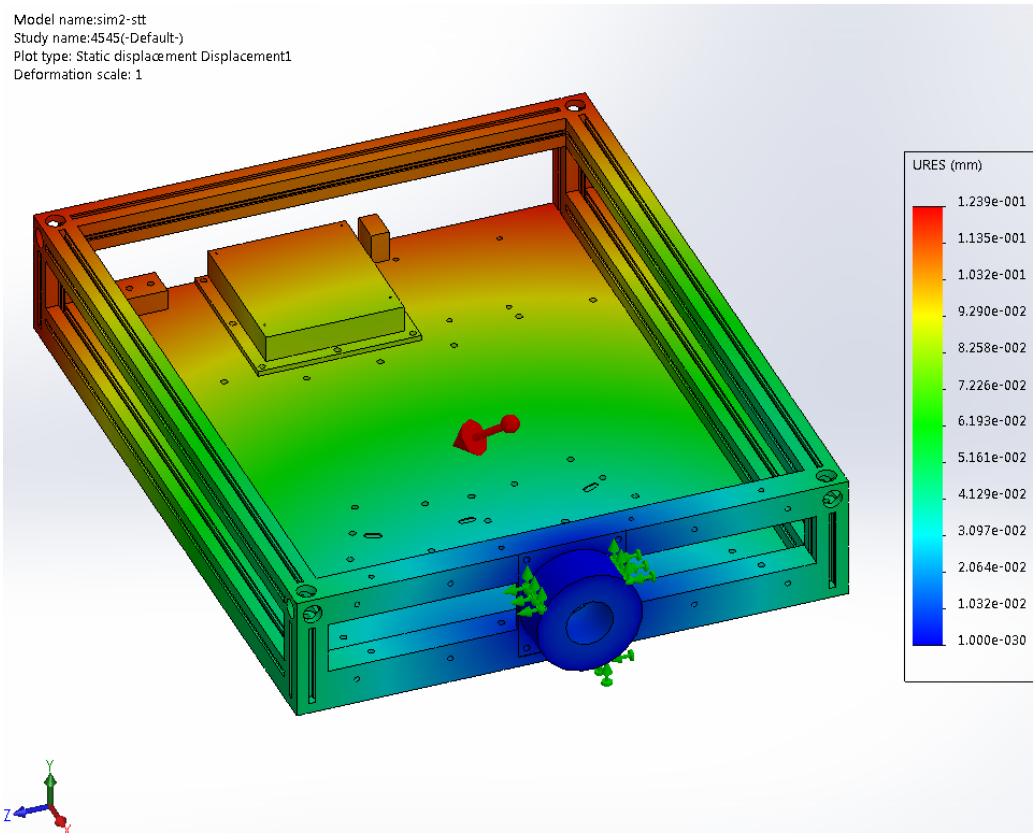


Figure D.27: *(δ,ϵ,ζ)=(45°,45°,45°).*

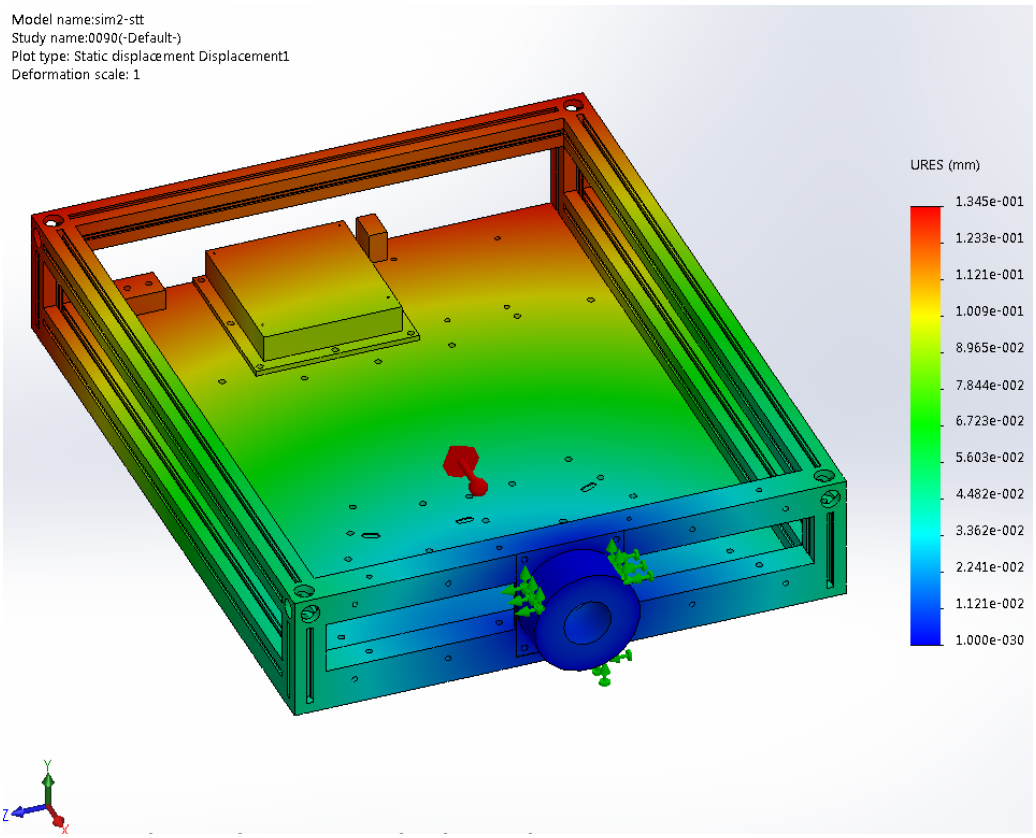


Figure D.28: *(δ,ϵ,ζ)=(0°,90°,90°).*

Appendix E

Estimated Material Cost of Production

In this appendix we present the estimated material cost of producing one COSAC unit. The products are described, and the quantities (Qtt.), estimated unit cost (EUC) and the estimated total cost (ETC) are listed, including VAT (23 %).

Product/Service	**Qtt.**	**EUC (€)**	**ETC (€)**
Mechanics			
Aluminum sheet, 350 mm×400 mm×25 mm	1	90.00	90.00
Aluminum sheet, 350 mm×400 mm×5 mm	1	45.00	45.00
Aluminum sheet, 500 mm×450 mm×0.75 mm	1	8.00	8.00
Anodization of the Produced Aluminum parts	1	82.00	82.00
Corner cubes, black anodized, 15 mm×15 mm×15 mm	4	1.25	5.00
Hexagonal Spacer M2 5 mm F/M	8	0.18	1.44
Hexagonal Spacer M3 5 mm Female/10 mm Male	4	0.25	1.00
Hexagonal Spacer M3 6 mm F/M	4	0.18	0,72
Linear slide carriage	2	13.25	26.50
Linear slide rail (300 mm) and carriage	1	21.50	21.50
Nut, M2.5	8	0.04	0.32
Nut, M3	51	0.04	2.04
OpenBeam, black anodized, 270 mm	5	3.00	15.00
OpenBeam, black anodized, 45 mm	4	0.50	2.00
Photopolymer Resin, black, 1 L	0.06	163.35	9.80
Screw Pozidriv, M2.5, 6 mm	12	0.05	0.60
Screw Pozidriv, M3, 5 mm	10	0.09	0.90
Screw Slot, M3, 10 mm	8	0.05	0.40
Screw Slot, M3, 12 mm	39	0.04	1.56
Screw Slot, M3, 20 mm	8	0.05	0.40
Self-locking nut, M3	103	0.05	4.64
Square headed bolt, M3, 12 mm	18	0.08	1.46
Square headed bolt, M3, 6 mm	66	0.06	3.66
Electronics			
Arduino Uno Rev. 3	1	23.75	23.75
Capacitor, 0.1 μF, C1206	4	0.20	0.8

Table E.1 continued from previous page

Product/Service	Qtt.	EUC (€)	ETC (€)
Capacitor, 10 μF, CT3528	4	0.60	2.4
Capacitor, 100 μF, D/7343-31R	1	2.00	2
Hamamatsu S1223	1	23.22	23.22
IEC C14 (male) 3 pin socket, 4 A	1	13.97	13.97
IVC102U, SOIC-14	1	15.50	15.5
LM1086, 5V, D2PACK	1	2.46	2.46
LTC2440IGN, SSOP-16	1	17.60	17.6
pIDDO-24bit v1.6 (or pIDDO-10bit v1.1) PCB	1	3.20	3.2
Potentiometer, 5 kΩ, 15 turn, RTRIMTS63Y	1	5.29	5.29
SD-Card Breakout Board, Adafruit	1	7.50	7.5
Stackable header kit for Arduino Uno	1	2.58	2.58
Power Supply			
IEC C13 (female) 3 pin plug, 6 A	1	2.40	2.4
Plastic box, 180 mm$\times$160 mm$\times$90 mm	1	8.00	8
Symmetrical 1 A Power Source Kit K8042	1	9.75	9.75
Thermoretractile Sleeve 1 m, $\varnothing$6.4 mm>3.2 mm	3	0.66	1.98
Toroid Transformer 230 V/24+24 V, 1.66+1.66 A, 80 VA	1	32.63	32.63
Optics			
DLP LightCrafter 3000	1	650.00	650
Lens, CSOPTLENS01 – 111-0222E	2	19.00	38
Lens, CSOPTLENS01 – 111-0210E	1	21.00	21
		Total	**1207.24**

Table E.1: *Estimated costs of materials and services for the production of a COSAC unit.*